ＮＰ２０００
取扱説明書

株式会社テクノランド

第３版　2003年05月10日

目次

1. ＣＰＵ基板・外観図 1
2. NP2000-OS の概要 3
3. 開発環境 4
3.1. C コンパイラ 4
3.2. NP2000 専用のアプリケーション開発・メンテナンス用ターミナル（NpTerm） 4
3.3. NP2000 専用の開発・メンテナンス用Ｉ／Ｆユニット(TLTERM2) 4
3.4. NP2000 専用画面作成支援ツール（SHTP.EXE） 4
3.5. NP2000 専用の SH 内蔵 FLASH メモリーへの OS ダウンロードツール（NｐLoader） 4
3.6. フォーマット変換ツール（BIN2S3.EXE） 4
3.7. NP2000 用標準フォントファイル 5
4. LCD 表示とタッチパネルの概要 6
5. NP2000 の起動 7
5.1. NP2000－OS 起動時の液晶画面 7
5.2. 起動時の開発・メンテナンスポートへの起動情報出力 8
5.3. デフォルト起動 8
6. OS モニターのコマンドと使用方法 10
6.1. GO コマンド：アプリケーションの実行 11
6.2. BREAK コマンド：アプリケーションの一時停止 11
6.3. STOP コマンド：アプリケーションの終了 11
6.4. DLINFO コマンド：ダウンロード情報の表示 11
6.5. AUTOEXEC コマンド：自動実行アプリケーションの設定・表示 11
6.6. AUTODISP コマンド：自動表示 BITMAP の設定・表示 12
6.7. MAN_EXEC コマンド：手動実行アプリケーションの設定・表示 12
6.8. BAUD コマンド：開発・メンテナンス用ポートのボーレート設定・表示 12
6.9. IPADR コマンド：IP ADDRESS の設定･表示 13
6.10. IPMSK コマンド： SUBNET MASK の設定･表示 13
6.11. IPCFG コマンド：IP 関係の設定を表示 13
6.12. VOL コマンド：液晶輝度の設定・表示 13
6.13. BL コマンド：液晶バックライトの ON/OFF・表示 14
6.14. DATE コマンド：日付の設定・表示 14
6.15. TIME コマンド：時刻の設定・表示 14
6.16. RTC コマンド：日付&時刻の設定・表示 15
6.17. TP コマンド：タッチパネル検出閾値の設定・表示 15
6.18. KB コマンド：キー押下時のブザー設定・表示 16
6.19. ERASE コマンド：ＦＬＡＳＨメモリーの消去・確認 16
6.20. JP コマンド：ジャンパーの状態を表示 17

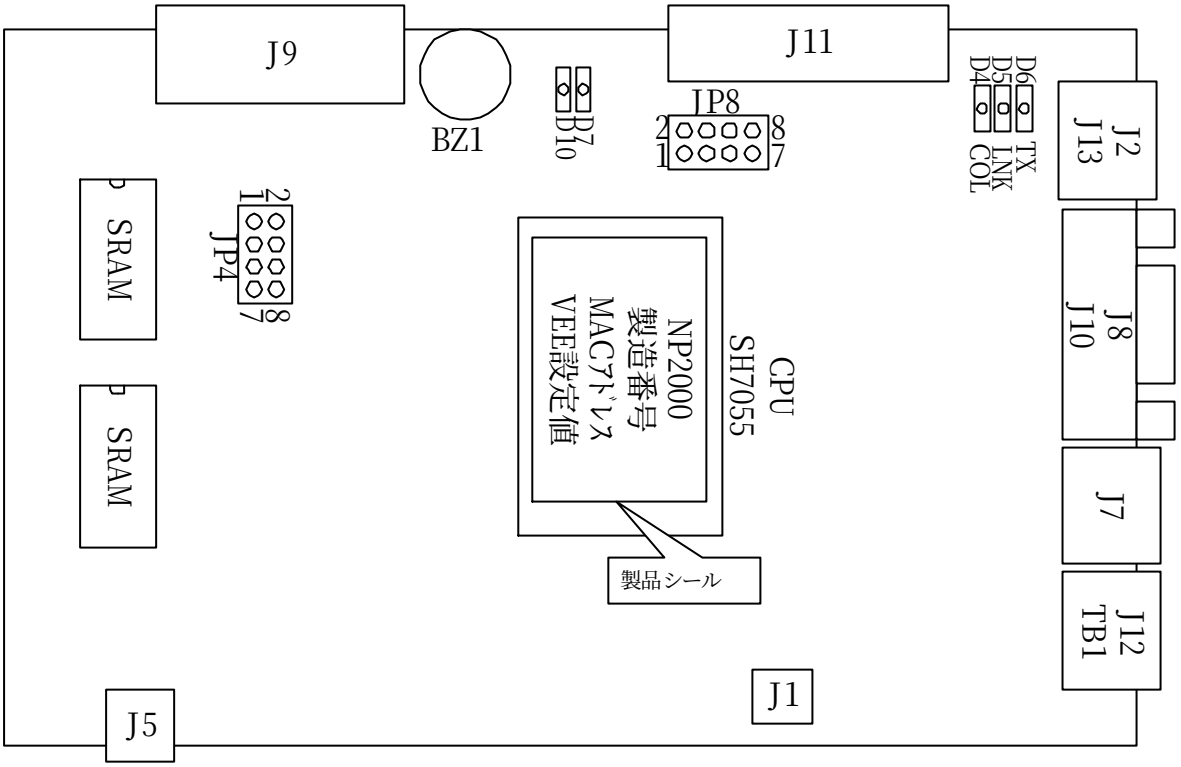
6.21. RESETコマンド：NP2000をリセットし再起動 17
6.22. Eコマンド：メモリーの変更 18
6.23. Dコマンド：メモリーのダンプ表示 18
6.24. WINBMPコマンド：Windows Bitmap の表示..... 19
6.25. ERRCHKコマンド：エラーログを表示 19
6.26. ERRCLRコマンド：エラーログを消去 20
6.27. ＬＢコマンド：バッテリー残量のチェック 20
6.28. STKCHKコマンド：スタックの使用状況を表示..... 20
6.29. HELP/?コマンド：OSモニターの使用方法を表示する。 20
7. アプリケーションの作成方法..... 21
7.1. NP2000-BIOSのセットアップ 21
7.1.1. NP2000用フォルダの作成 21
7.1.2. NP2000-BIOSのコピー 21
7.1.3. NP2000-BIOSファイルの解凍 21
7.2. サンプルアプリケーション用フォルダの作成 23
7.3. サンプルプログラムのソースファイル作成 23
7.4. LINKサブ・コマンドファイルの作成 23
7.5. 環境変数設定用BATファイル（SETENV.BAT）の作成..... 24
7.6. MSDOSプロンプトでの操作 26
7.6.1. ドライブ及びディレクトリの移動..... 26
7.6.2. 環境変数設定用BAT(SETENV.BAT)の実行..... 26
7.6.3. サンプルプログラムのコンパイル..... 27
7.6.4. サンプルプログラムのリンク..... 27
7.6.5. サンプルプログラムのファイル変換..... 28
7.6.6. MSDOSプロンプトの終了..... 30
7.7. サンプルプログラムのダウンロード 31
7.7.1. アプリケーション領域の消去..... 31
7.7.2. HELLO.MOTのダウンロード..... 32
7.8. アプリケーションの実行 34
7.9. アプリケーション作成上の注意事項..... 35
7.9.1. C言語ソースプログラムについて 35
7.9.2. メモリマップ及びセクションについて..... 35
8. システムコール 39
8.1. システム関連システムコール..... 39
8.1.1. 【関数名】wait 39
8.1.2. 【関数名】GetTickCount 39
8.1.3. 【関数名】Buzzer..... 39
8.1.4. 【関数名】mk_crc 39

8.1.5. 【関数名】err_crc 40
8.1.6. 【関数名】RtcRead 40
8.1.7. 【関数名】RtcWrite 40
8.1.8. 【関数名】LedOut 41
8.1.9. 【関数名】SysReset 41
8.1.10. 【関数名】BackupDataSave 41
8.1.11. 【関数名】BackupDataLoad 42
8.1.12. 【関数名】BatChk 42
8.2. ＬＣＤ＆タッチパネル関連システムコール 43
8.2.1. 【関数名】VOLcont 43
8.2.2. 【関数名】LcdBackLight 43
8.2.3. 【関数名】CursorSetup 43
8.2.4. 【関数名】LcdPuts 43
8.2.5. 【関数名】LcdPuts2 44
8.2.6. 【関数名】ModifyLcdAttr 45
8.2.7. 【関数名】LcdClr 45
8.2.8. 【関数名】LcdNormal 45
8.2.9. 【関数名】LcdReverse 45
8.2.10. 【関数名】SetupGamen 45
8.2.11. 【関数名】isTpHit 46
8.2.12. 【関数名】GetTp 46
8.2.13. 【関数名】KeyBuzzer 46
8.2.14. 【関数名】isTpHit_XY 46
8.2.15. 【関数名】GetTp_XY 47
8.2.16. 【関数名】GetTp_Tbl 47
8.2.17. 【関数名】LcdButton 47
8.2.18. 【関数名】LcdPrintf 49
8.2.19. 【関数名】GetTpTable 49
8.2.20. 【関数名】SetTpTable 50
8.3. ＲＳ－ＣＯＭＭ関連システムコール 51
8.3.1. 【関数名】RsCommOpen 51
8.3.2. 【関数名】RsCommClose 52
8.3.3. 【関数名】RsCommCtrl 52
8.3.4. 【関数名】RsCommStat 53
8.3.5. 【関数名】RsCommGetch 53
8.3.6. 【関数名】RsCommPutch 54
8.3.7. 【関数名】RsCommKbHit 55
8.3.8. 【関数名】RsCommGets 55

8.3.9. 【関数名】RsCommPuts 57
8.3.10. 【関数名】RsCommRead 57
8.3.11. 【関数名】RsCommWrite 58
8.3.12. 【関数名】RsCommPrintf 59
8.3.13. 【関数名】Rs485Init 60
8.3.14. 【関数名】Rs485RcvCtrl 60
8.3.15. 【関数名】Rs485DrvCtrl 61
8.3.16. 【関数名】printd 61
8.3.17. 【関数名】ercd_print 61
8.3.18. 【関数名】DebugPrintf 61
8.4. ＴＣＰ関連システムコール 63
8.4.1. 【関数名】TcpSocket 63
8.4.2. 【関数名】TcpAccept 63
8.4.3. 【関数名】TcpConnect 64
8.4.4. 【関数名】TcpIsConnect 64
8.4.5. 【関数名】TcpClose 65
8.4.6. 【関数名】TcpSocketClose 65
8.4.7. 【関数名】TcpSocketCloseAll 66
8.4.8. 【関数名】TcpRead 66
8.4.9. 【関数名】TcpWrite 67
8.4.10. 【関数名】TcpGetch 67
8.4.11. 【関数名】TcpIsRcv 68
8.4.12. 【関数名】TcpGets 68
8.4.13. 【関数名】TcpPutch 69
8.4.14. 【関数名】TcpPuts 70
8.4.15. 【関数名】TcpPrintf 70
8.4.16. 【関数名】TcpPush 71
8.4.17. 【関数名】TcpFlush 71
8.4.18. 【関数名】SetIpaddr 71
8.4.19. 【関数名】GetIpaddr 72
8.5. グラフィック関連システムコール 74
8.5.1. 【関数名】DotCheck 74
8.5.2. 【関数名】DotSet 74
8.5.3. 【関数名】Line 74
8.5.4. 【関数名】LineStyle 74
8.5.5. 【関数名】Circle 75
8.5.6. 【関数名】BoxFill 75
8.5.7. 【関数名】PutImg 76

8.5.8. 【関数名】PutImgOr 76
8.5.9. 【関数名】GetImg 76
8.5.10. 【関数名】PutWinbmp 76
8.5.11. 【関数名】TextOut 77
8.6. ＩＯポート関連システムコール 78
8.6.1. 【関数名】IniPort 78
8.6.2. 【関数名】OutPort 78
8.6.3. 【関数名】InPort 79
8.6.4. 【関数名】InAD 79
9. 利用可能マクロ 80
9.1. 組込みタイマー 80
9.1.1. TIMER_L 80
9.1.2. TIMER_1 80
9.1.3. TIMER_2 80
9.1.4. TIMER_SEC 80
10. 注意事項 81
10.1. ファイル送信に関する注意事項 81
10.1.1. ファイル送信について 81
10.1.2. ファイル（データ）格納領域 81
10.1.3. ファイル(データ)格納領域の消去 82
10.1.4. ダウンロード情報 82
10.1.5. ファイル比較 82
10.1.6. 送信ファイルの作成 82
10.2. Windowsビットマップに関する注意事項 83
10.3. ＯＳアップデート回数の制限 83

1. ＣＰＵ基板・外観図



● コネクタ

コネクタ番号	コネクタ名称	備考
J9	開発･メンテナンス用ポート	TLTERM2 を介して開発･メンテナンス用パソコンと接続します。
J11	拡張 I/O コネクタ	
J2	LAN(10base-T)コネクタ	RJ-45
J10	RS232C(COM1)コネクタ	DSUB9ﾋﾟﾝ(ﾌﾟﾗｸﾞ)
J7	RS485(COM2)コネクタ	全 2 重／半 2 重はプログラマブル
J12	電源コネクタ	＋５Ｖ電源供給用コネクタ
J1	外付け電池用コネクタ	円筒型リチウム電池接続用コネクタ（※１）
J14	電池ホルダ	コイン型リチウム電池用電池ホルダ（※１）
J5	液晶バックライト インバータ用コネクタ	標準では、インバータケーブルの NP2000 側は半田付けしていますので、J5 は実装されていません。

(※１)　標準では、コイン型リチウム電池を使用します。この場合は、J1 に外付け電池は接続できません。出荷時オプションにより、外付けリチウム電池、内蔵電気２重層コンデンサを指定することが出来ます。出荷時オプションにつきましては、「NP2000 製品仕様書」をご覧下さい。

● ＬＥＤ

LED 番号	LED 名称	説明
D7	ﾓﾆﾀｰ LED1	動作モニター用ＬＥＤです。メインタスクのループ中はD7、D10共に点滅しています。アプリケーションプログラムからは、LedOut関数で制御可能です。
D10	ﾓﾆﾀｰ LED0	
D4	COL	LAN(10Base-T)でコリジョンが発生していることを示すＬＥＤです。
D5	LNK	LAN(10Base-T)のリンクテストが成功したことを示すＬＥＤです。
D6	TX	LAN(10Base-T)に送信中であることを示すＬＥＤです。

● ジャンパー

JP4、JP8共にシステムで使用しています。
設定を変えずにご使用ください。

● 製品シール

製品シールには、製造番号、ＭＡＣアドレス（Ｅｔｅｒｎｅｔアドレス）、ＶＥＥ（液晶駆動電圧）設定値が記入されています。

2. NP2000-OS の概要

NP2000-OSは、iTron4仕様準拠のマルチタスクOSです。

NP2000-OSは、起動時に標準で以下のタスクを生成・実行します。

- ■ メインタスク
- ■ ＯＳモニター制御タスク
- ■ タッチパネル制御タスク
- ■ ＬＣＤ表示制御タスク
- ■ ブザー制御タスク
- ■ ＴＣＰ／ＩＰ制御タスク

アプリケーションの実行開始は、メインタスク上で監視され、自動実行、手動実行及びGOコマンドによるアプリケーション実行を検出した場合に、ユーザープログラム用タスクを生成・開始しメインタスク自身はスリープ状態に移行することによりユーザープログラム用タスクに制御を渡します。
ユーザータスクは所定の番地に格納されたアプリケーションプログラムを呼び出すことによりアプリケーションを実行します。

アプリケーションプログラム終了時は、ユーザータスクは自タスクを終了させ、メインタスクを起床させることによりメインタスクに制御を戻します。

STOP コマンドは OS モニター制御タスクによって処理され、ユーザータスクを強制的に終了させ、メインタスクを起床することによりメインタスクに制御を戻させます。

起床されたメインタスクは、アプリケーションに開放していたリソースをすべて解放後、ユーザータスクを削除しアプリケーションの実行開始を監視する状態に戻ります。

3. 開発環境

NP2000上で動作する、アプリケーションプログラムを開発するためには、Ｃ言語のソースファイルのコンパイル、リンク等を行うＣコンパイラと、作成した実行ファイル（モトローラＳフォーマット）をダウンロードするための専用ターミナルプログラムが必要になります。

また、パソコンのCOMポートとNP2000の開発・メンテナンス用ポートを接続するために、RS232Cストレートケーブル（モデムケーブル）及びNP2000専用の開発・メンテナンス用Ｉ／Ｆユニット(TLTERM2)が必要になります。

3.1. Ｃコンパイラ

対応するＣコンパイラは以下の通りです。

日立マイクロコンピュータ
開発環境システム
SuperH RISC engine
C/C++ complier Pakage
Ver 5.1C
型名：Ｐ０７００ＣＡＳ５－MWR

3.2. NP2000専用のアプリケーション開発・メンテナンス用ターミナル（NpTerm）

NpTermはNP2000専用のアプリケーション開発･メンテナンス用ターミナルです。
詳細につきましては、「NpTerm操作説明書」をご覧下さい。

3.3. NP2000専用の開発・メンテナンス用Ｉ／ＦユニットTLTERM2)

TLTERM2はNP2000の開発･メンテナンス用ポートの信号レベルをRS232C信号レベルに変換するとともに、ユニット上のＳＷでCPU(SH7055)の起動モードを制御し、NP2000の開発･メンテナンスを行うためのユニットです。

詳細につきましては、「TLTERM2取扱説明書」をご覧下さい。

3.4. NP2000専用画面作成支援ツール（SHTP.EXE）

SHTP.EXEはNP2000専用のタッチパネル画面開発支援ツールです。
操作方法につきましては、「SHTP操作説明書」をご覧下さい。

3.5. NP2000専用のSH内蔵FLASHメモリーへのOSダウンロードツール（Ｎｐ Loader）

NpLoaderはNP2000専用のSH7055内蔵Flashメモリーへの NP2000-OSダウンロードツールです。
詳細につきましては、「NpLoader操作説明書」をご覧下さい。

3.6. フォーマット変換ツール（BIN2S3.EXE）

BIN2S3.EXEは汎用のフォーマット変換ツールです。

NP2000 にファイル（データ）をダウンロードする場合に、NP2000 にダウンロードできるモトローラＳフォーマットのファイルに変換するために使用します。
BIN2S3.EXE の使用方法につきましては、「BIN2S3 操作説明書」をご覧下さい。

ファイルのダウンロードは、NpTerm を使用して行います。この詳細に関しましては、「NpTerm 操作説明書」をご覧下さい。

3.7. NP2000 用標準フォントファイル

NP2000 の液晶表示に用いる標準フォントファイルです。
圧縮ファイル ”NPFNT101.LZH ” に格納されています。（ファイル名末尾の「101」はバージョンを示します。バージョンが異なる場合は読み替えてください。）
ファイルを解凍すると以下の 3 つの標準フォントファイル（モトローラＳ形式）が出来ます。

① NPZENFNT.ABS 標準全角フォント
② NPHANFNT.ABS 標準半角フォント
③ NPKEIFNT.ABS 標準罫線フォント

出荷時は、FLASH メモリーのフォント領域に書き込まれていますので、通常は上記ファイルを書き込む必要はありません。

「ERASE」コマンドで、誤ってフォント領域を消去した場合には、NpTerm の「ファイル送信」を使用して上記フォントをすべてダウンロード し直して下さい。

4. LCD 表示とタッチパネルの概要

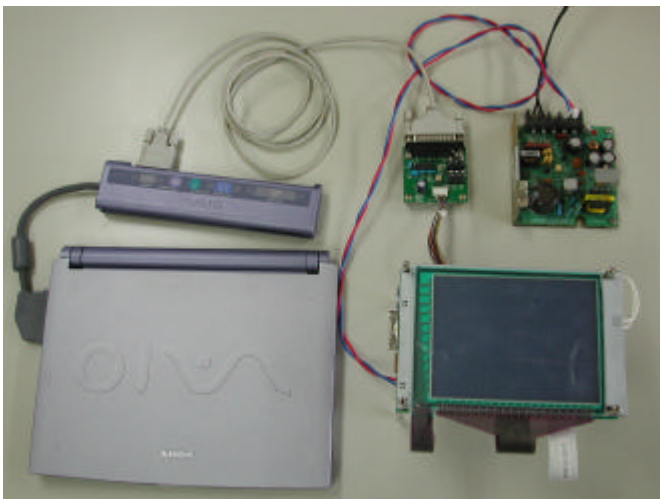
液晶表示は３２０ドット×２４０ラインの２値モノクロ液晶でそのエリアに２０×１２の分解能を持つタッチパネルで構成されています。従って NP2000-OS はタッチパネルの分解能にあわせた４０×１２のキャラクタ表示を基本とします。

キャラクタ座標は左上が（１，１）で、右下が（４０，１２）とする座標系となります。
従って表示できる文字数は、半角文字の場合で４０文字×１２行、全角文字の場合で２０文字×１２行となります。

また、グラフィック系のシステムコールは、キャラクタ座標とは異なるグラフィック座標を使用します。
グラフィック座標は左上が（０，０）で、右下が（３１９，２３９）とする座標系となります。

5. NP2000 の起動

① 下の写真のように、開発･メンテナンス用パソコン、TLTERM2、NP2000 及び＋５V 電源を接続します。

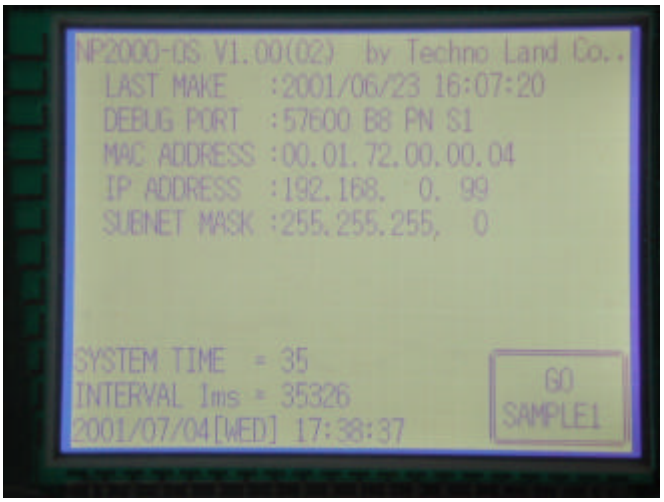


② パソコンの電源を投入し、Windows が立ち上がった後に、NpTerm を起動します。

③ NP2000 の電源（＋５V 電源）を投入し、NP2000 を起動します。

5.1. NP2000－OS 起動時の液晶画面

NP2000 が起動すると、NP2000-OS の起動画面が、液晶に表示されます。



起動画面では以下の項目を表示します。

- NP2000-OS のバージョン及び作成日時
- デバッグポートの通信設定
- Ethernet の MAC アドレス
- IP アドレス及びサブネット・マスク
- システムタイマー及び時計表示

GO SAMPLE1

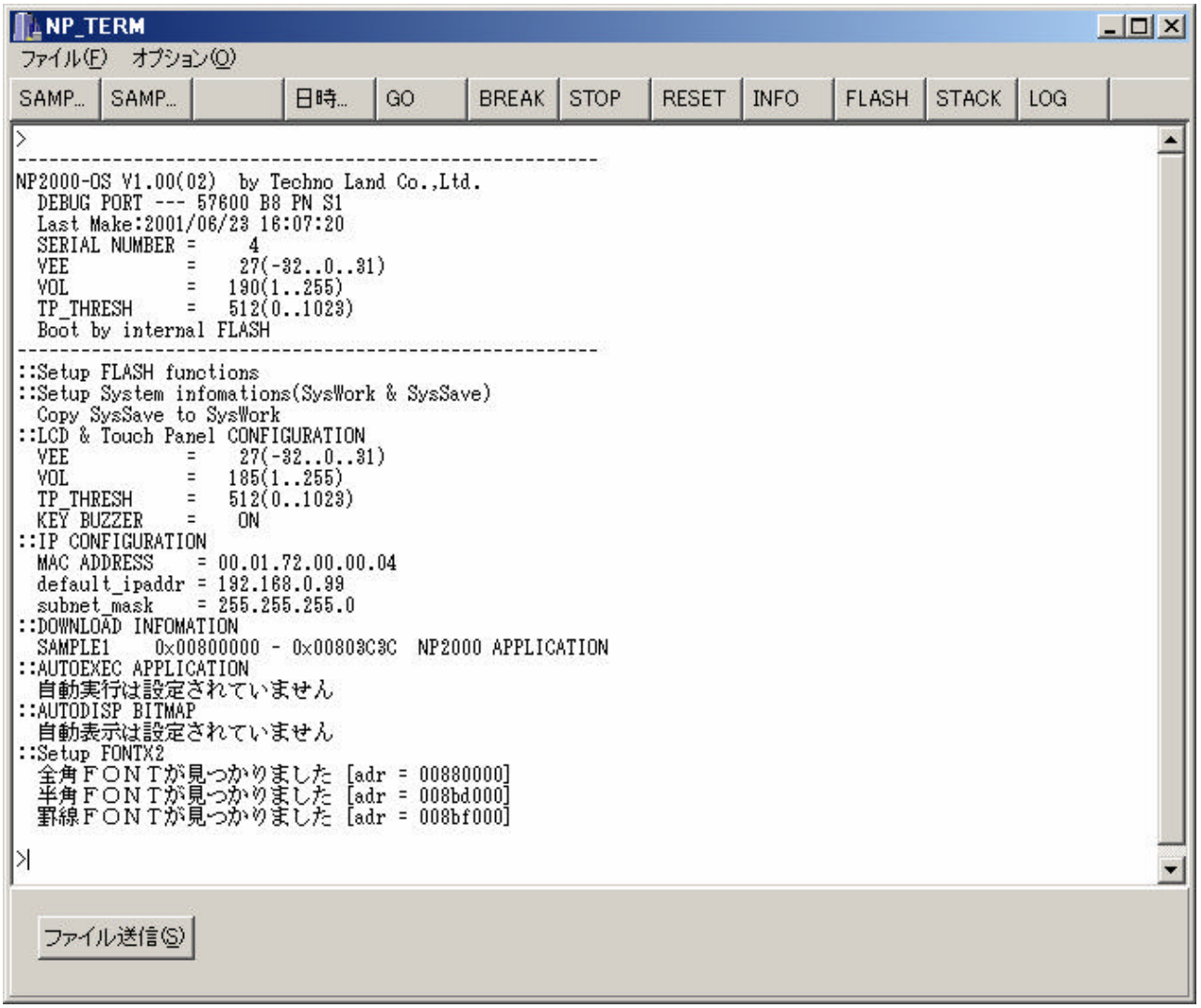
ボタンは、「MAN_EXEC」コマンドで手動実行アプリケーションが「SAMPLE1」を設定されている場合に表示されます。
この状態でこの部分を、押下すると「SAMPLE1」が起動します。

初期状態及び「MAN_EXEC」コマンドで手動実行アプリケーションを「OFF」にしている場合は、この部分に上記ボタンは表示されません。

※ 自動起動のアプリケーションが設定されている場合は、NP2000-OS 起動後すぐに設定されたアプリケーションを実行するため、NP2000-OS 起動画面は表示されません。

5.2. 起動時の開発・メンテナンスポートへの起動情報出力

NP2000 起動時に、起動情報が開発･メンテナンス用ポートに出力されます。
この情報は、NpTerm 上に表示されます。



※ NpTerm を後から起動した場合は、起動時の情報は表示されません。
この場合は、NP2000 を再起動することにより起動情報を表示することが出来ます。

5.3. デフォルト起動

TLTERM2 の「SH1」スイッチを「0」側に倒して起動すると、NP2000-OS は下記の項目を設定に関係なくデフォルト値で起動します。

① 開発・メンテナンス用ポートの通信速度
「BAUD」コマンドによる設定に関係なく、開発･メンテナンス用ポートの通信速度を 19200bps で起動します。

② IP アドレス及びサブネットマスク
「IPADR」、「IPMSK」コマンドによる設定に関係なく、

IP ｱﾄﾞﾚｽ=192.168.0.99

サブネットマスク=255.255.255.0

で起動します。

③ 自動実行アプリケーション、自動表示ビットマップ

「AUTOEXEC」、「AUTODISP」コマンドによる設定に関係なく、自動実行アプリケーションも、自動表示ビットマップも「無し」で起動します。

④ 初期の輝度設定

「VOL」コマンドによる設定に関係なく、NP2000-0S ダウンロード時に NpLoader の「液晶輝度設定」の項で設定した輝度設定で起動します。

⑤ タッチパネル検出時の入力閾値

「TP」コマンドによる設定に関係なく、NP2000-0S ダウンロード時に NpLoader の「タッチパネル入力スレッシュ」の項で設定した入力閾値で起動します。

6. OS モニターのコマンドと使用方法

「２．NP2000-OSの概要」で記したように、NP2000-OSはマルチタスクＯＳであり、標準のタスクとして「ＯＳモニター制御タスク」が常時動作しています。

「ＯＳモニター制御タスク」は、NP2000 の開発･メンテナンス用ポートの入力を常時監視しており、このポートに入力があった場合は、コマンド解析を行い有効なコマンドであった場合はＯＳモニターへのコマンドとして実行します。

■ 設定値に関する注意事項

NP2000 では、各種設定値を FLASH メモリー内に保存し、起動時にこの FLASH メモリー内の設定値を SRAM 上にコピーして実際の設定に使用します。

OS モニターのコマンド実行時に、設定値として２種類表示される場合がありますが、SysSaveの記述がある設定値は、FLASH メモリー内に保存された設定値を示し、SysWork の記述がある設定値は SRAM 上にコピーされた設定値を示します。

コマンドによっては、SysSave 側の設定値のみを変更し、現在の動作には反映させない場合があります。

この場合は、SysWork 設定値と SysSave 設定値は値が異なり、変更は次回起動時から有効になります。

例）

開発・メンテナンス用ポートのボーレート設定(BAUD コマンド)

57600baudで起動

```
>baud<Enter>
DEBUG PORT BAUDRATE(SysWork) = 57600[baud]
DEBUG PORT BAUDRATE(SysSave) = 57600[baud]
```

19200baudに変更

```
>baud 19200<Enter>
システム領域消去中
Flash[0]:Sector[eH]:adr=008e0000H:Erase OK
SYSTEM 情報 書き込み中
書き込み完了
SYSTEM 情報 CRC 書き込み中
書き込み完了 CRC = 0xD1A1
DEBUG PORT BAUDRATE(SysWork) = 57600[baud]
DEBUG PORT BAUDRATE(SysSave) = 19200[baud]
  設定は次回起動時から有効になります。
>
```

設定値(19200)は、FLASHメモリーに書き込まれ、SysSave側は変更が反映されていますが、SysWork側は変更前の57600のままで変更されていないことがわかります。

6.1. GOコマンド：アプリケーションの実行

■ 書式１　GO [アプリケーション名]<Enter>

[アプリケーション名]で指定された、プログラムをダウンロードリストから検索し、ダウンロードされている場合は実行します。

但し、すでにアプリケーション実行中の場合は、本コマンドは受け付けられません。

■ 書式２　GO<Enter>(BREAKコマンドでアプリケーションが停止中の場合)

停止中のアプリケーションの実行を再開します。

■ 書式３　GO<Enter>(MAN_EXECコマンドにより手動実行アプリケーションが設定されている場合)

MAN_EXECコマンドで設定されたアプリケーションを実行します。

6.2. BREAKコマンド：アプリケーションの一時停止

■ 書式　BREAK<Enter>

実行中のアプリケーションを停止させます。

再開はGOコマンド、終了はSTOPコマンドを使用します。

6.3. STOPコマンド：アプリケーションの終了

■ 書式　STOP<Enter>

実行中または停止中のアプリケーションを強制的に終了させます。

6.4. DLINFOコマンド：ダウンロード情報の表示

■ 書式　DLINFO<Enter>

NpTermを使用してダウンロードしたデータの情報を表示します。

6.5. AUTOEXECコマンド：自動実行アプリケーションの設定・表示

NP2000はAUTOEXECコマンドで自動実行アプリケーションを設定されている場合は、OS起動後直ちに設定されているアプリケーションをダウンロードリストから検索し、ダウンロードされている場合は自動的に実行します。

■ 書式１　AUTOEXEC<Enter>

自動実行機能の設定状況を表示します。

■ 書式２　AUTOEXEC OFF<Enter>

自動実行機能をＯＦＦにします。

■ 書式３　AUTOEXEC [アプリケーション名]

自動実行させるアプリケーション名を設定します。

※設定は次回起動時から有効になります。

6.6. AUTODISP コマンド：自動表示 BITMAP の設定・表示

NP2000 は AUTODISP コマンドで自動表示 BITMAP を設定されている場合は、OS 起動後直ちに設定されている BITMAP をダウンロードリストから検索し、ダウンロードされている場合は自動的に表示します。

■ 書式１　AUTODISP<Enter>

自動表示 BITMAP の設定状況を表示します。

■ 書式２　AUTODISP OFF<Enter>

自動表示機能をＯＦＦにします。

■ 書式３　AUTODISP [BITMAP 名]

自動表示させる BITMAP 名を設定します。

※設定は次回起動時から有効になります。

6.7. MAN_EXECコマンド：手動実行アプリケーションの設定・表示

NP2000 は MAN_EXEC コマンドで手動実行アプリケーションを設定されている場合は、GO<Enter>のみで設定されているアプリケーションをダウンロードリストから検索し、ダウンロードされている場合は自動的に実行します。

MAN_EXEC コマンドで手動実行アプリケーションを設定されている場合は、OS の LCD 画面右下に手動実行ボタンが表示されます。

このボタンを押下することでも、手動実行アプリケーションを実行することができます。

■ 書式１　MAN_EXEC<Enter>

手動実行アプリケーションの設定状況を表示します。

■ 書式２　MAN_EXEC OFF<Enter>

手動実行機能をＯＦＦにします。

■ 書式３　MAN_EXEC [アプリケーション名]

指定されたアプリケーションを手動実行アプリケーションに設定します。

6.8. BAUD コマンド：開発・メンテナンス用ポートのボーレート設定・表示

■ 書式１　BAUD<Enter>

開発・メンテナンス用ポートのボーレート設定を表示します。

■ 書式２　BAUD [ボーレート設定]

開発・メンテナンス用ポートのボーレートを指定されたボーレートに設定します。
設定できるボーレートは 2400、4800、9600、19200、38400、57600[BAUD]のいずれかです。
※設定は次回起動時から有効になります。

6.9. IPADR コマンド：IP ADDRESS の設定･表示

■ 書式１　IPADR<Enter>

NP2000 に設定されている IP アドレスを表示します。

■ 書式２　IPADR [IP アドレス]

指定された IP アドレスを、NP2000 の IP アドレスとして設定します。
※設定は次回起動時から有効になります。

※ NP2000 の IP アドレス及びサブネットマスクは、SetIpaddr 関数、GetIpaddr 関数を使用して、アプリケーションプログラムからの設定・確認を行うことも可能です。

6.10. IPMSK コマンド： SUBNET MASK の設定･表示

■ 書式１　IPMSK<Enter>

NP2000 に設定されている SUBNET MASK を表示します。

■ 書式２　IPMSK [SUBNET MASK]<Enter>

指定された SUBNET MASK を、NP2000 の SUBNET MASK として設定します。
※設定は次回起動時から有効になります。

※ NP2000 の IP アドレス及びサブネットマスクは、SetIpaddr 関数、GetIpaddr 関数を使用して、アプリケーションプログラムから設定・確認を行うことも可能です。

6.11. IPCFG コマンド：IP 関係の設定を表示

■ 書式　IPCFG<Enter>

NP2000 の TCP/IP 関係の設定（MAC アドレス、IP アドレス、SUBNET MASK）を表示します。

6.12. VOL コマンド：液晶輝度の設定・表示

■ 書式１　VOL<Enter>

現在の輝度設定を表示します。

■ 書式２　VOL [輝度設定]<Enter>

NP2000 起動時の輝度を設定します。
設定範囲は１～２５５で、コマンド実行時に直ちに輝度及び輝度の初期値を変更します。
数値が小さくなると表示は濃くなり、大きくなると薄くなります。

標準の輝度は１９０です。

※ 輝度は VOLcont 関数でも設定できます。この場合は、その時点での輝度は変更できますが起動時の輝度は変更されません。

6.13. BL コマンド：液晶バックライトの ON/OFF・表示

■ 書式１　BL<Enter>

液晶バックライトの現在の状態を表示します。

■ 書式２　BL ON<Enter>

液晶バックライトを ON にします。

■ 書式３　BL OFF<Enter>

液晶バックライトを OFF にします。

※ 液晶バックライトは、LcdBackLight 関数によりアプリケーションプログラムからも制御することが出来ます。

6.14. DATE コマンド：日付の設定・表示

■ 書式　DATE<Enter>

NP2000 内蔵時計の日付を設定・表示します。

上記コマンドを入力すると、以下のように現在の日付を表示するとともに新しい日付を入力するように求められます。

```
Current date is THU 2001-06-21
Enter new date (yyyy-mm-dd):
```

ここで、日付の確認の場合は<Enter>のみを入力しコマンドを終了させます。

日付を変更する場合は、”2001-6-22”のように、西暦-月-日の順で’-‘で区切って入力してください。

6.15. TIME コマンド：時刻の設定・表示

■ 書式　TIME<Enter>

NP2000 内蔵時計の時刻を設定・表示します。

上記コマンドを入力すると、以下のように現在の時刻を表示するとともに新しい時刻を入力するように求められます。

```
Current time is 13:55:15
Enter new date (hh:mm:ss):
```

ここで、時刻の確認の場合は<Enter>のみを入力しコマンドを終了させます。

時刻を変更する場合は、”15:30:55”のように、時-分-秒の順で’:‘で区切って入力してください。

6.16. RTC コマンド：日付＆時刻の設定・表示

■ 書式 RTC<Enter>

NP2000 内蔵時計の日付及び時刻を設定・表示します。

上記コマンドを入力すると、以下のように現在の日付・時刻を表示するとともに新しい日付・時刻を入力するように求められます。

```
Current rtc is WED 2001-06-20 01:07:58
Enter new rtc (yyyy-mm-dd hh:mm:ss):
```

ここで、日付・時刻の確認の場合は、<Enter>のみを入力しコマンドを終了させます。

日付・時刻を変更する場合は、”2001-06-21 17:08:30”のように、日付は西暦-月-日の順で”-“で区切って入力し、時刻は時-分-秒の順で’:‘で区切って入力して下さい。

日付と時刻の間は半角スペースで区切ってください。

※ 日付・時刻は RtcRead 関数、RtcWrite 関数を使用してアプリケーションプログラムから設定・確認を行うことも可能です。

6.17. TP コマンド：タッチパネル検出閾値の設定・表示

NP2000 では、タッチパネルの押下検出を、CPU 内蔵の 10 ビット A/D コンバータで行っています。
このため、タッチパネル未押下／押下の判定を行うための検出閾値を設定する必要があります。
TP コマンドは、この検出閾値の設定・表示を行うコマンドです。

■ 書式１ TP<Enter>

タッチパネル検出閾値の設定を表示します。

以下に示すような設定情報が表示されます。

```
>TP
タッチパネル検出情報の表示 = OFF
タッチパネル検出閾値(SysWork.tp_thresh) = 512
タッチパネル検出閾値(SysSave.tp_thresh) = 512
設定範囲: 1..1023
```

２種類の閾値が表示されますが、これは現在の設定値(SysWork)と起動時の設定値(SysSave)を示します。(「設定値に関する注意事項」参)

検出閾値設定については、通常は同一の値が表示されます。

■ 書式２ TP [検出閾値]<Enter>

タッチパネル検出閾値を設定します。

設定範囲は、１～１０２３です。

数値が小さくなるほど検出閾値が下がり、検出しにくくなります。

数値を大きくすると、軽く触れただけでも検出するようになりますが、大きくしすぎるとタッチパネルを押下していなくてもキーを検出してしまいます。

デフォルトの検出閾値は、５１２です。

■ 書式３　TP ON<Enter>

タッチパネル検出情報の表示機能を有効にします。

この機能は、タッチパネル制御タスクから直接検出情報を開発・メンテナンス用ポートに出力する機能です。

アプリケーションプログラムが、デバッグ情報を pirntd 関数等で開発・メンテナンス用ポートに出力しているような場合は、競合が発生しアプリケーション及びタッチパネル検出の反応が低下する場合があります。

従って、この機能はタッチパネルのチェック時等のメンテナンス時にのみ使用してください。

■ 書式４　TP　OFF<Enter>

前述の、タッチパネル検出情報の表示機能を無効にします。

通常はこの状態で使用してください。

起動時は、タッチパネル検出情報の表示機能は無効(OFF)に設定してあります。

6.18. KB コマンド：キー押下時のブザー設定・表示

■ 書式１　KB<Enter>

キー(タッチパネル)押下時のブザー設定を表示します。

■ 書式２　KB [ON/OFF]

キー押下時のブザーを設定します。

ON に設定すると、キー押下検出時にブザーが鳴ります。

OFF に設定すると、キー押下を検出してもブザーは鳴りません。

※ キー押下時のブザー設定は KeyBuzzer 関数を使用してアプリケーションプログラムから設定することも可能です。

6.19. ERASE コマンド：ＦＬＡＳＨメモリーの消去・確認

■ 書式１　ERASE INFO<Enter>

FLASH メモリーの消去情報及びダウンロード情報を表示します。

```
>ERASE INFO
Flash[0]:Sector[0H]:adr=00800000H:Erase NG --- APPLICATION
Flash[0]:Sector[2H]:adr=00820000H:Erase OK --- APPLICATION
Flash[0]:Sector[4H]:adr=00840000H:Erase OK --- APPLICATION
Flash[0]:Sector[6H]:adr=00860000H:Erase NG --- USER
Flash[0]:Sector[8H]:adr=00880000H:Erase NG --- FONT
Flash[0]:Sector[aH]:adr=008a0000H:Erase NG --- FONT
```

```
Flash[0]:Sector[cH]:adr=008c0000H:Erase OK --- RESERVE
Flash[0]:Sector[eH]:adr=008e0000H:Erase NG --- SYSTEM
-- DOWNLOAD INFOMATION --
  SAMPLE2     0x00800000 - 0x00806FBC  NP2000 APPLICATION     [CRC OK]
```

ERASE NG と表示されている領域には、NpTerm からのダウンロードまたはプログラム上からの書き込みにより何らかのデータが書き込まれています。

■ 書式２　ERASE APP<Enter>

FLASH メモリーのアプリケーション領域を消去します。

FLASH メモリーはデータの上書きが出来ない為、アプリケーションプログラムをダウンロードする場合は、必ずダウンロードする前にアプリケーション領域を消去してください。

■ 書式３　ERASE FONT<Enter>

標準 FONT が格納されている領域を消去します。

誤って、この領域を消去すると LCD への文字表示が出来なくなります。

通常は使用しないでください。

■ 書式４　ERASE [番地] <Enter>

消去する領域の先頭番地(16 進表記)を指定して FLASH メモリーの消去を行います。

３ブロックあるアプリケーション領域の１ブロックのみを消去するような場合に使用できます。

但し、誤った番地を指定して、意図しない領域を消去してしまう可能性もあるため、使用に際しては十分注意してください。

6.20. JP コマンド：ジャンパーの状態を表示

■ 書式　JP<Enter>

ジャンパーの状態を表示します。

```
>jp
SH_JP1  is OFF(DEFAULT JUMPER)      (SH_JP1 は、JP8 の 1-2 ピン間を示します)
SH_JP2  is OFF                      (SH_JP2 は、JP8 の 3-4 ピン間を示します)
SH_JP3  is OFF                      (SH_JP3 は、JP8 の 5-6 ピン間を示します)
SH_JP4  is OFF                      (SH_JP4 は、JP8 の 7-8 ピン間を示します)
PLD_JP1 is ON                       (PLD_JP1 は、JP4 の 1-2 ピン間を示します)
PLD_JP2 is OFF                      (PLD_JP2 は、JP4 の 3-4 ピン間を示します)
```

6.21. RESETコマンド：NP2000 をリセットし再起動

■ 書式　RESET<Enter>

NP2000 をソフト的にリセットし再起動します。

6.22. Eコマンド：メモリーの変更

■ 書式１　E [ADDRESS] <Enter>

バイト(8ビット)単位でメモリーの変更を行います。
アドレス及びデータは全て１６進数で入力してください。
操作方法は以下の通りです。
①２桁の数値が入力されると自動的にメモリーを変更し、次のアドレスに移ります。
②１桁＋<Enter>を入力された場合もメモリーを変更し、次のアドレスに移ります。
③半角スペース＋<Enter>を入力すると、メモリーの内容は変更せずに次のアドレスに移ります。
④<Enter>のみを入力するとEコマンドを終了します。

■ 書式２　EW [ADDRESS] <Enter>

ワード(16ビット)単位でメモリーの変更を行います。
アドレス及びデータは全て１６進数で入力してください。
操作方法は以下の通りです。
①４桁の数値が入力されると自動的にメモリーを変更し、次のアドレスに移ります。
②３桁以下の数値＋<Enter>を入力された場合もメモリーを変更し、次のアドレスに移ります。
③半角スペース＋<Enter>を入力すると、メモリーの内容は変更せずに次のアドレスに移ります。
④<Enter>のみを入力するとEコマンドを終了します。

■ 書式３　ED [ADDRESS] <Enter>

ダブルワード(32ビット)単位でメモリーの変更を行います。
アドレス及びデータは全て１６進数で入力してください。
操作方法は以下の通りです。
①８桁の数値が入力されると自動的にメモリーを変更し、次のアドレスに移ります。
②７桁以下の数値＋<Enter>を入力された場合もメモリーを変更し、次のアドレスに移ります。
③半角スペース＋<Enter>を入力すると、メモリーの内容は変更せずに次のアドレスに移ります。
④<Enter>のみを入力するとEコマンドを終了します。

※ アドレスを入力せずEコマンドを実行した場合は、直前のEコマンドの終了番地を引き継ぎます。
※ メモリーの変更はＳＲＡＭに対してのみ有効です。ＳＲＡＭ以外の領域に対して Eコマンドを使用しないでください。

6.23. Dコマンド：メモリーのダンプ表示

■ 書式１　D [ADDRESS]<Enter>

バイト(8ビット)単位でメモリーのダンプ表示を行います。
画面右側には、ASCII文字での表示を行います。
制御コード等で表示ができない文字に関しては空白を表示します。

アドレスは１６進数で入力してください。

■ 書式２　DJ [ADDRESS] <Enter>

バイト(8ビット)単位でメモリーのダンプ表示を行います。
Dコマンドとの違いは、画面右側の文字表示部が日本語（Shift-JIS）に対応している点です。
アドレスは１６進数で入力してください。

■ 書式３　DW [ADDRESS] <Enter>

ワード(16ビット)単位でメモリーのダンプ表示を行います。
ダンプ表示のみで、画面右側の文字表示は行いません。
アドレスは必ず2N（２の倍数）で指定してください。
アドレスは１６進数で入力してください。

■ 書式４　DD [ADDRESS] <Enter>

ダブルワード(32ビット)単位でメモリーのダンプ表示を行います。
ダンプ表示のみで、画面右側の文字表示は行いません。
アドレスは必ず4N（４の倍数）で指定してください。
アドレスは１６進数で入力してください。

※ アドレスを入力せずDコマンドを実行した場合は、直前のDコマンドの終了番地を引き継ぎます。

6.24. WINBMPコマンド：Windows Bitmap の表示

■ 書式　WINBMP X Y [bitmap] Z<Enter>

予め、NpTermでダウンロードしてあるWindowsBitmapを表示します。
(X,Y)で表示する左上座標を指定します。
Zは表示方法を指定します。(Z=0：反転表示、Z=1:通常表示)

※ Windows-Bitmapは白黒画像のみ対応しています。
※ 表示するWindows-BitmapはあらかじめBIN2S3.EXEを使用して、モトローラSフォーマットに変換した上でNpTermによりダウンロードしておいて下さい。
※ ファイル変換時の注意事項につきましては、「10.2　Windowsビットマップに関する注意事項」の項をご参照願います。
※ (X,Y)グラフィック座標で指定してください。(X=0..319, Y=0..239)

6.25. ERRCHKコマンド：エラーログを表示

■ 書式　ERRCHK<Enter>

OS内部のエラーログ保存領域に保存されているエラーの一覧を表示します。

※ エラーログは、ercd_print関数を使用した場合に保存されます。詳細については、ercd_print関数の説明をご参照願います。

6.26. ERRCLR コマンド：エラーログを消去

■ 書式　ERRCLR<Enter>

OS 内部のエラーログ保存領域に保存されているエラーの一覧を消去します。

6.27. ＬＢコマンド：バッテリー残量のチェック

■ 書式　LB<Enter>

バッテリー残量の状態を表示します。

「バッテリー残量が減っています」と表示された場合はバッテリーを交換して下さい。

6.28. STKCHK コマンド：スタックの使用状況を表示

■ 書式　STKCHK<Enter>

スタックの使用状況を表示します。

スタックの未使用領域は unused として表示されていますが、これは各タスクの初期化時に書き込んでおいた値から変化していない部分のサイズであり参考値です。

6.29. HELP/?コマンド：OS モニターの使用方法を表示する。

■ 書式１　HELP<Enter>

■ 書式２　?<Enter>

OS モニターの簡単な使用方法を表示します。

7. アプリケーションの作成方法

本項では、簡単なサンプルプログラムを作成して、実際にNP2000上で動作させるまでの手順を説明します。アプリケーションの作成に先立って、あらかじめ「３．開発環境」に従って、必要な開発環境をセットアップして下さい。

日立純正 C/C++コンパイラにつきましては、本書ではコマンドラインですべて操作するように説明をしております。

統合環境をご使用になられる場合は、日立純正 C/C++コンパイラのマニュアルを参照して行って下さい。

また、日立純正 C/C++コンパイラは”C:\Hew”にインストールされているものとして説明しておりますので、インストールされているフォルダが異なる場合は、インストールされているフォルダに合わせて読み替えていただきますようお願いいたします。

※ 日立純正C/C++コンパイラの使用方法の詳細に関しましては、コンパイラ付属の説明書をご覧下さい。

7.1. NP2000-BIOSのセットアップ

NP2000-BIOSは、”D:\NP2000”フォルダにセットアップするものとして説明しております。

セットアップするドライブまたはフォルダ名が異なる場合は、セットアップされるフォルダに合わせて読み替えていただきますようお願いいたします。

7.1.1. NP2000用フォルダの作成

エクスプローラ等で、Dドライブに NP2000用フォルダ ”D:\NP2000”を作成します。

7.1.2. NP2000-BIOSのコピー

作成したフォルダにNP2000-BIOSファイルをコピーします。

NP2000-BIOSファイルは、”V1_00_02.LZH”のようにバージョン名が付けられた圧縮ファイルです。

7.1.3. NP2000-BIOSファイルの解凍

解凍ツールを使用して、NP2000-BIOSファイルを解凍してください。

解凍によって、NP2000用フォルダが以下の構成になっていることを確認してください。

<D:\NP2000>　　NP2000用フォルダ
　<BIOS>　　ヘッダー、ライブラリ及びスタートアップコード
　<OS>　　NP2000-OS実行ファイル
　<SAMPLE1>　サンプルプログラム
　<SAMPLE2>　サンプルプログラム

※ 解凍ツールは別途ご用意ください。

● <BIOS>フォルダについて

当フォルダには以下の3種類のファイルが格納されています。

① ヘッダーファイル

アプリケーションプログラムから、システムコールや NP2000 用ライブラリコールを行うための各種宣言や設定を記述した C 言語用ヘッダーファイルが格納されています。

NP2000.hはすべてのヘッダーファイルをインクルードしていますので、通常は C 言語ソースの先頭で

```
#include “NP2000.h”
```

として、NP2000.h のみをインクルードして使用します。

② NP2000 用ライブラリファイル（NP2K.LIB）

システムコールだけでは実現できない関数については NP2000 用ライブラリ（NP2K.LIB）で実装しています。

リンク時には必ず、NP2K.LIB をリンクするようにしてください。

- C 言語標準関数を使用するためには、日立純正 C/C++コンパイラに付属の shcnpic.lib もリンクする必要があります。

具体的には「7.4 LINK サブ・コマンドファイルの作成」の項をご参照願います。

③ スタートアップ用ファイル

スタートアップ用オブジェクトファイル（START.OBJ、INIT.OBJ）及びそれらを作成するためのソースファイル、MAKE ファイル、BAT ファイルです。

- START.ASM　スタートアップコード（アセンブラソース）
- START.OBJ　START.ASM のアセンブル実行後のオブジェクトファイル
- INIT.C　スタートアップ後の初期化コード（C言語ソース）
- INIT.OBJ　INIT.C のコンパイル実行後のオブジェクトファイル
- STARTUP.MAK　START.OBJ、INIT.OBJ 作成用の MAKE ファイル（Borland MAKE用）
- WINMAKE1.BAT　MAKE 実行用の BAT ファイル

リンク時には必ず、START.OBJ、INIT.OBJ、アプリケーションプログラムの OBJ の順でリンクしてください。

具体的には「7.4 LINK サブ・コマンドファイルの作成」の項をご参照願います。

※ 通常はスタートアップ用オブジェクトファイルは、START.OBJ、INIT.OBJ をこのままご使用下さい。

※ ソースファイルも添付していますので変更は可能ですが、変更する場合は「7.9 アプリケーション作成上の注意事項」およびコンパイラのマニュアル等を参考に注意深く行ってください。

● <OS>フォルダについて

当フォルダには以下の 2 つのファイルが格納されています。

① NP2KOS.MOT　NP2000-OS の実行ファイル（モトローラS形式）

② History.pdf　変更履歴

出荷時は、NP2KOS.MOT は CPU(SH7055)内蔵 Flash メモリーに書き込まれていますので、出荷時バージョンのままであれば上記ファイルを使用する必要はありません。

ＯＳアップデート時は、NpLoaderを使用して、NP2KOS.MOTを SH7055 内蔵 Flash メモリーにダウンロードします。

NpLoader及びＯＳアップデート方法の詳細については、「NpLoader操作説明書」をご覧下さい。

※ NP2000-OS アップデート時は<BIOS>フォルダの内容も更新する必要があります。
<BIOS>フォルダと NP2000-OS のバージョンが一致していないと作成したアプリケーションが正常に動作しない場合がありますのでご注意願います。

7.2. サンプルアプリケーション用フォルダの作成

エクスプローラ等を使用して、NP2000用フォルダ”D:\NP2000”下にフォルダ”Hello”を作成します。
これ以後に作成する、HELLO.C、HELLO.SUB、SETENV.BAT等はすべてこの”D:\NP2000\Hello\”に保存してください。

7.3. サンプルプログラムのソースファイル作成

以下の通りＣ言語のソースファイル（HELLO.C）を作成します。

```
----ここから----
#define COM     /* */
#include "np2000.h"
void main(void)
{
    T_COMPO_BUTTON exit_btn  = {NULL, 0, 31, 10, 10, 3, 2, LCD_REVERSE, "サンプル\n終     了", 0,
0x1b};

    LcdButton(&exit_btn);
    LcdPrintf(14, 6, LCD_NORMAL, "HELLO WORLD");

    do {
        if (isTpHit())  break;
    } while(1);

    GetTp();
}
----ここまで----
```

これを d:\NP2000\Hello\HELLO.Cとして保存します。

7.4. LINK サブ・コマンドファイルの作成

以下の通りリンカに対するサブ･コマンドファイル(HELLO.SUB)を作成します。

----ここから----

```
PRINT hello
INPUT ..\BIOS\start.obj ..\BIOS\init.obj
INPUT hello.obj
OUTPUT hello
LIBRARY ..\BIOS\np2k.lib
LIBRARY C:\Hew\Tools\Hitachi\Sh\5_1_0\Lib\shcnpic
NODEBUG
ROM (D,R)
START APP_ID(00800000),P,C,D(00800100)
START BBIOS(00478000)
START R,B,BWORK(00480000)
START BBUPMEM(004B0000)
FORM A
EXIT
```

----ここまで----

これを d:\NP000\Hello\HELLO.SUB として保存します。

※ 上記リストで赤字で示した部分は、コンパイラをインストール先に合わせて変更してください。

※ リンカ（リンケージ･エディタ）に対するサブコマンドの詳細は日立純正 C/C++コンパイラのマニュアルをご覧下さい。

7.5. 環境変数設定用 BAT ファイル（SETENV.BAT）の作成

以下の通り環境変数を設定するための BAT ファイル(SETENV.BAT)を作成します。

----ここから----

```
@ECHO OFF
path = C:\Hew\Tools\Hitachi\Sh\5_1_0\bin
set SHC_LIB=c:\Hew\Tools\Hitachi\Sh\5_1_0\bin
set SHC_INC=c:\Hew\Tools\Hitachi\Sh\5_1_0\include
set SHC_TMP=c:\Hew\Tmp
```

----ここまで----

これを d:\NP000\Hello\SETENV.BAT として保存します。

※ 上記リストで赤字で示した部分は、コンパイラのインストール先に合わせて変更してください。

※ 日立純正 C/C++コンパイラをインストールした状態では、”c:\Hew\Tmp”は作成されていません。
エクスプローラ等を使用してフォルダを作成するか、別のテンポラリフォルダを指定してください。

有効なテンポラリフォルダが指定されていない場合は、コンパイル時にエラーが発生してコンパイル出来ませんのでご注意ください。

※ 環境変数 SHC_LIB は、日立純正 C/C++コンパイラの ”bin” フォルダを指定します。
標準ライブラリの格納フォルダではありませんのでご注意ください。

※ 環境変数 SHC_LIB、SHC_INC、SHC_TMP の詳細につきましては、コンパイラのマニュアルをご覧下さい。

7.6. MSDOS プロンプトでの操作

Windows98 の「スタート」－「ﾌﾟﾛｸﾞﾗﾑ」－「MS-DOS プロンプト」を選択し、MSDOS プロンプトを起動します。

（WindowsMe の場合は、「スタート」－「ﾌﾟﾛｸﾞﾗﾑ」－「ｱｸｾｻﾘ」－「MS-DOS プロンプト」を選択します。）

以下に、MS-DOS プロンプト（DOS 窓）での実際の操作を示しながら順次手順の説明を行います。

青字で示した部分が、プロンプトに対してキーボードから入力した内容です。

7.6.1. ドライブ及びディレクトリの移動

ドライブ指定及び CD コマンドを実行して、ドライブ及びディレクトリをサンプルアプリケーション用フォルダ”D:\NP2000\Hello”に移動します。

```
C:\>D:<Enter>

D:\>CD \NP2000\HELLO<Enter>

D:\NP2000\Hello>DIR<Enter>

 ドライブ D: のボリュームラベルは DATA
 ボリュームシリアル番号は 0E24-19DD
 ディレクトリは D:\NP2000\Hello

.              <DIR>         01-07-06  17:25 .
..             <DIR>         01-07-06  17:25 ..
HELLO    C             334   01-07-06  18:08 hello.c
HELLO    SUB           368   01-07-06  17:59 hello.sub
SETENV   BAT           175   01-07-06  17:48 setenv.bat
         3 個                   877 バイトのファイルがあります.
         2 ﾃﾞｨﾚｸﾄﾘ       8,816.10 MB の空きがあります.
```

7.6.2. 環境変数設定用 BAT(SETENV.BAT)の実行

あらかじめ作成しておいた、環境変数設定用 BAT コマンド”SETENV.BAT”を実行します。

```
D:\NP2000\Hello>SETENV<Enter>
```

ファイル先頭で「@ECHO OFF」によって、エコー出力を OFF に設定しているため、実行結果は表示されません。

「SET」コマンドで、環境変数の設定を確認します。

```
D:\NP2000\Hello>SET<Enter>
COMSPEC=C:\WINDOWS\COMMAND.COM
TEMP=C:\WINDOWS\TEMP
TMP=C:\WINDOWS\TEMP
winbootdir=C:\WINDOWS
windir=C:\WINDOWS
BLASTER=A220 I5 D1 T4 P330
```

```
PROMPT=$p$g
SYS=e:
VZDEF=VZJ
CMDLINE=vzibmj -Z
PATH=C:\HEW\TOOLS\HITACHI\SH\5_1_0\BIN
SHC_LIB=c:\Hew\Tools\Hitachi\Sh\5_1_0\bin
SHC_INC=c:\Hew\Tools\Hitachi\Sh\5_1_0\include
SHC_TMP=c:\Hew\Tmp
```

最後の４行に”SETEMV.BAT”で設定した、環境変数が設定されていることが確認できます。

7.6.3. サンプルプログラムのコンパイル

あらかじめ作成しておいた、HELLO.Cをコンパイルします。

```
D:\NP2000\Hello>SHC -cp=sh2 -op=1 -i=..\BIOS HELLO.C<Enter>
```

プログラムに異常があった場合は、エラーまたは警告のメッセージが表示されます。
コンパイルが正常に終了した場合は、何も表示されません。
「DIR」コマンドによって、ディレクトリの内容を確認します。

```
D:\NP2000\Hello>DIR<Enter>

 ドライブ D: のボリュームラベルは DATA
 ボリュームシリアル番号は 0E24-19DD
 ディレクトリは D:\NP2000\Hello

.            <DIR>         01-07-06  17:25 .
..           <DIR>         01-07-06  17:25 ..
HELLO    C              334  01-07-06  18:08 hello.c
HELLO    SUB            368  01-07-06  17:59 hello.sub
SETENV   BAT            175  01-07-06  17:48 setenv.bat
HELLO    OBJ            627  01-07-06  18:37 HELLO.obj
         4 個              1,504 バイトのファイルがあります.
         2 ﾃﾞｨﾚｸﾄﾘ       8,816.09 MB の空きがあります.
```

コンパイラによって、オブジェクトファイル”HELLO.OBJ”が生成されていることが確認できます。

7.6.4. サンプルプログラムのリンク

あらかじめ作成しておいたLINKサブコマンド・ファイルHELLO.SUBを使用してリンクを行います。

```
D:\NP2000\Hello>LNK /su=HELLO.SUB<Enter>
H SERIES LINKAGE EDITOR Ver. 6.0E
Copyright (C) Hitachi, Ltd.1989,1996
Copyright (C) Hitachi ULSI Systems Co., Ltd. 1990,1996
Licensed Material of Hitachi, Ltd.

:     PRINT hello
```

```
:       INPUT ..\BIOS\start.obj ..\BIOS\init.obj
:       INPUT hello.obj
:       OUTPUT hello
:       LIBRARY ..\BIOS\np2k.lib
:       LIBRARY C:\Hew\Tools\Hitachi\Sh\5_1_0\Lib\shcnpic
:       NODEBUG
:       ROM (D,R)
:       START APP_ID(00800000),P,C,D(00800100)
:       START BBIOS(00478000)
:       START R,B,BWORK(00480000)
:       START BBUPMEM(004B0000)
:       FORM A
:       EXIT

LINKAGE EDITOR COMPLETED
```

“LINKAGE EDITOR COMPLETED”のメッセージでリンカが正常に終了したことが確認できます。

「DIR」コマンドで、ディレクトリの内容を確認します。

```
D:\NP2000\Hello>DIR<Enter>

 ドライブ D: のボリュームラベルは DATA
 ボリュームシリアル番号は 0E24-19DD
 ディレクトリは D:\NP2000\Hello

.              <DIR>         01-07-06  17:25 .
..             <DIR>         01-07-06  17:25 ..
HELLO    C             334   01-07-06  18:08 hello.c
HELLO    SUB           368   01-07-06  17:59 hello.sub
SETENV   BAT           175   01-07-06  17:48 setenv.bat
HELLO    OBJ           627   01-07-06  18:37 HELLO.obj
HELLO    ABS        23,119   01-07-06  18:37 hello.ABS
HELLO    MAP        18,267   01-07-06  18:37 hello.MAP
         6 個              42,890 バイトのファイルがあります.
         2 ﾃﾞｨﾚｸﾄﾘ       8,816.05 MB の空きがあります.
```

リンカによって、”HELLO.ABS”と”HELLO.MAP”の２つのファイルが生成されていることが確認できます。

7.6.5. サンプルプログラムのファイル変換

リンカが出力したABS形式のファイルをモトローラS形式のファイルに変換します。

```
D:\NP2000\Hello>CNVS HELLO.ABS HELLO.MOT<Enter>
H SERIES SYSROF STYPE OBJECT CONVERTER Ver. 2.0A
Copyright (C) Hitachi, Ltd. 1988,1997
Copyright (C) HITACHI MICROCOMPUTER SYSTEM LTD. 1988,1997
Licensed Material of Hitachi, Ltd.


OBJECT CONVERTER COMPLETED
```

“OBJECT CONVERTER COMPLETED”のメッセージでファイル形式の変換が正常に終了したことが確認できます。
「DIR」コマンドで、ディレクトリの内容を確認します。

```
D:\NP2000\Hello>DIR<Enter>

 ドライブ D: のボリュームラベルは DATA
 ボリュームシリアル番号は 0E24-19DD
 ディレクトリは D:\NP2000\Hello

.            <DIR>          01-07-06  17:25 .
..           <DIR>          01-07-06  17:25 ..
HELLO    C              334  01-07-06  18:08 hello.c
HELLO    SUB            368  01-07-06  17:59 hello.sub
SETENV   BAT            175  01-07-06  17:48 setenv.bat
HELLO    OBJ            627  01-07-06  18:37 HELLO.obj
HELLO    MOT         44,282  01-07-06  18:38 HELLO.MOT
HELLO    ABS         23,119  01-07-06  18:37 hello.ABS
HELLO    MAP         18,267  01-07-06  18:37 hello.MAP
         7 個             87,172 バイトのファイルがあります.
         2 ﾃﾞｨﾚｸﾄﾘ      8,816.00 MB の空きがあります.
```

ファイル変換プログラムによって、モトローラＳ形式のファイル”HELLO.MOT”が生成されたことが確認できます。
この”HELLO.MOT”が、NP2000の実行ファイルとなります。

拡張子(ABS)は、モトローラS形式のファイルに適用する場合もあります。
この為、NpTermの「ファイル送信」時のファイル選択画面では、「.ABS」「.MOT」を標準の拡張子として扱っています。
しかし、上記の”HELLO.ABS”はモトローラＳ形式ではないため、NpTermから「ファイル送信」は出来ません。
従って、間違いを避けるためフォーマット変換前の”HELLO.ABS”は削除しておきます。

```
D:\NP2000\Hello>DEL HELLO.ABS<Enter>

D:\NP2000\Hello>DIR<Enter>

 ドライブ D: のボリュームラベルは DATA
 ボリュームシリアル番号は 0E24-19DD
 ディレクトリは D:\NP2000\Hello

.            <DIR>          01-07-06  17:25 .
..           <DIR>          01-07-06  17:25 ..
HELLO    C              334  01-07-06  18:08 hello.c
HELLO    SUB            368  01-07-06  17:59 hello.sub
SETENV   BAT            175  01-07-06  17:48 setenv.bat
HELLO    OBJ            627  01-07-06  18:37 HELLO.obj
```

```
HELLO    MOT        44,282  01-07-06  18:38 HELLO.MOT
HELLO    MAP        18,267  01-07-06  18:37 hello.MAP
         6 個              64,053 バイトのファイルがあります.
         2 ﾃﾞｨﾚｸﾄﾘ      8,816.02 MB の空きがあります.
```

7.6.6. MSDOS プロンプトの終了

以上でサンプルプログラムは完成しましたので、MSDOS プロンプトを終了します。

```
D:\NP2000\Hello>EXIT<Enter>
```

7.7. サンプルプログラムのダウンロード

本項では、NpTerm を使用してサンプルプログラム実行ファイル”HELLO.C”のダウンロードの手順について説明します。
あらかじめ「5. NP2000 の起動」を参考にして、NpTerm 及び NP2000 を起動しておいて下さい。
以下に、NpTerm 画面での実際の操作を示しながら順次手順の説明を行います。
青字で示した部分が、プロンプトに対してキーボードから入力した内容です。

7.7.1. アプリケーション領域の消去

アプリケーションをダウンロードするためには、ダウンロードするためのアプリケーション領域が消去されている必要があります。
消去情報の取得には「ERASE INFO」コマンドを使用します。

```
>ERASE INFO<Enter>
Flash[0]:Sector[0H]:adr=00800000H:Erase NG --- APPLICATION
Flash[0]:Sector[2H]:adr=00820000H:Erase OK --- APPLICATION
Flash[0]:Sector[4H]:adr=00840000H:Erase OK --- APPLICATION
Flash[0]:Sector[6H]:adr=00860000H:Erase NG --- USER
Flash[0]:Sector[8H]:adr=00880000H:Erase NG --- FONT
Flash[0]:Sector[aH]:adr=008a0000H:Erase NG --- FONT
Flash[0]:Sector[cH]:adr=008c0000H:Erase OK --- RESERVE
Flash[0]:Sector[eH]:adr=008e0000H:Erase NG --- SYSTEM
-- DOWNLOAD INFOMATION --
  HELLO       0x00800000 - 0x00803C3C  NP2000 APPLICATION     [CRC OK]
```

この例では、消去情報の最上段と末尾のダウンロード情報から、アプリケーション領域にはアプリケーションプログラム”HELLO”が格納されていることが確認できます。
先ほど作成したプログラムと同一名のため、このまま「ファイル送信」を行うとファイルの比較機能が働きますので、ここでは「ERASE APP」コマンドを使用してアプリケーション領域を消去します。

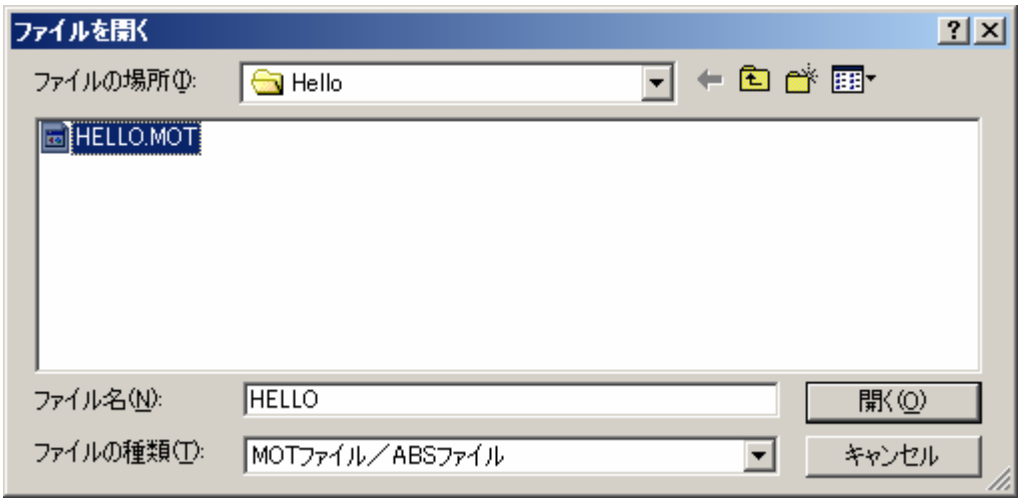
```
>ERASE APP<Enter>
アプリケーション領域消去中
Flash[0]:Sector[0H]:adr=00800000H:Erase OK
Flash[0]:Sector[2H]:adr=00820000H:Erase OK
Flash[0]:Sector[4H]:adr=00840000H:Erase OK
delete HELLO - NP2000 APPLICATION
ダウンロード情報を書き換えます！
システム領域消去中
Flash[0]:Sector[eH]:adr=008e0000H:Erase OK
SYSTEM 情報 書き込み中
書き込み完了
SYSTEM 情報 CRC 書き込み中
書き込み完了 CRC = 0xE958
```

以上で、アプリケーション領域の消去は完了しました。

7.7.2. HELLO.MOTのダウンロード

前項でアプリケーション領域は消去されていますので、サンプルプログラム HELLO.MOT をダウンロードすることが出来ます。

NpTerm の「ファイル送信」ボタンをクリックすると、下図の「ファイルを開く」画面が開きます。



ここで、先ほど作成した HELLO.MOT を選択し、「開く」ボタンをクリックすると、下図の「データ送信中」画面が開き送信状況をバー表示します。

送信が完了すると、「データ送信中」画面は閉じて、NpTerm のメイン画面に戻ります。

この時、NpTerm のターミナル画面上に、以下のような NP2000 からのダウンロード完了メッセージが表示されます。

```
>
終了レコード受信
ダウンロード正常終了
ダウンロード情報格納開始
システム領域消去中
Flash[0]:Sector[eH]:adr=008e0000H:Erase OK
SYSTEM 情報 書き込み中
書き込み完了
SYSTEM 情報 CRC 書き込み中
書き込み完了 CRC = 0xBDAF
NAME  = HELLO
TYPE  = NP2000 APPLICATION
START = 0x00800000
```

```
LAST   = 0x00803C3C
SIZE   = 15421[BYTE]
CRC    = 0x4214
```

これで、サンプルプログラムはアプリケーション領域に、アプリケーション実行プログラム HELLO として格納されました。

7.8. アプリケーションの実行

アプリケーションの実行は「GO」コマンドにより行います。

```
>GO HELLO<Enter>
アプリケーション [HELLO ]を実行します

>GO APPLICATION
Search HELLO from Download Infomation
DATA TYPE = NP2000 Application
Hit HELLO in Download Infomation
Check CRC ---- OK
HELLO を実行します
```

「GO HELLO<Enter>」に対して、OS 内部でアプリケーション実行プログラム HELLO の検索、CRC チェックを行った後に HELLO を実行していることが確認できます。

HELLO 実行により、NP2000 の液晶画面表示は下図の様になります。

サンプル 終　　了	ボタンで終了します。

終了すると、NpTerm のターミナル画面上に下記のような終了メッセージが表示されます。

```
：リソース開放－開始
TcpSocketCloseAll():START
TcpSocketCloseAll():END
：リソース開放－完了
HELLO を終了しました

>
```

以上で、サンプルプログラム HELLO は終了し、液晶画面が再び OS 起動画面に戻ります。

7.9. アプリケーション作成上の注意事項

7.9.1. C言語ソースプログラムについて

すべてのアプリケーションプログラム用ソースファイルに、NP2000.hをインクルードしてください。
また、ソースファイルの1つのみ（通常はmain()関数が含まれるソースファイル）では、NP2000.hをインクルードする前に、キーワードCOMをコメントアウトする定義文を記述してください。

● main()関数の含まれるソース

```
#define COM      /* */
#include “NP2000.h”
```

● その他のソースファイル

```
#include “NP2000.h”
```

キーワードCOMは、グローバル変数の定義を制御しています。
グローバル変数は、COM をコメントアウトしたファイルで定義され、その他のファイルではキーワード COM はexternに置き換えられ、外部参照となります。

7.9.2. メモリマップ及びセクションについて

「7.4 LINKサブ・コマンドの作成」で作成した、HELLO.SUBによって、サンプルプログラムHELLOのメモリーマップおよびセクション構成は下表の通り構成されます。

物理番地	デバイス	セクション	説明
0x00478000-0x0047FFFF	RAM(SRAM)	BBIOS	NP2000-OS、アプリケーション共有領域
0x00480000-0x004AFFFF	RAM(SRAM)	R, B, BWORK	アプリケーション用SRAM領域
0x004B0000-0x004BFFFF	RAM(SRAM)	BBUPMEM	アプリケーション用バックアップメモリ領域
0x00800000-0x008000FF	ROM(FLASH)	APP_ID	NP2000用アプリケーション識別子領域
0x00800100-0x0085FFFF	ROM(FLASH)	P, C, D	アプリケーション用プログラム、データ領域

メモリマップを定義するのは、HELLO.SUB中の以下の部分です。

```
ROM (D,R) ---------------------------------- ①
START APP_ID(00800000),P,C,D(00800100) ------- ②
START BBIOS(00478000) ----------------------- ③
START R,B,BWORK(00480000)
START BBUPMEM(004B0000) --------------------- ④
```

この部分につきましては、通常の場合は特に変更する必要はありません。
変更される場合は、以下の説明をご参考にされて、注意深く変更されますようお願いいたします。

① ROM化支援機能の指定

サブコマンド”ROM (D,R)”はリンカに、ROM化支援機能を指定しています。

これにより、初期化領域の初期値をセクション D に格納し、実際の初期化変数はセクション R に確保されます。

アプリケーションプログラム起動時にセクション D からセクション R へ初期値をコピーする必要がありますが、これは INIT.C:APP_INITSCT()関数で行っています。(INIT.C は<BIOS>フォルダに格納されています。)

② アプリケーションプログラム用 ROM(FLASH メモリー)内のメモリー・マップの指定

サブコマンド " START APP_ID(00800000),P,C,D(00800100)"はリンカに ROM(FLASH メモリー)のセクション先頭アドレス及び結合順序の指定を行っています。

セクション APP_ID は、NP2000 用のアプリケーションであることを示す識別情報を格納した領域です。
識別情報は、START.ASM で設定しています。(START.ASM は<BIOS>フォルダに格納されています。)
NP2000-OS はアプリケーションプログラムのダウンロード時に、先頭部分を調べて、このアプリケーションプログラム識別情報を検出した場合は、ダウンロードデータがアプリケーションプログラムであることを認識し、その情報（データタイプ）をダウンロード情報に格納しています。
ダウンロード情報表示時（DLINFO コマンド、ERASE INFO コマンド）でデータのタイプが NP2000 APPLICATION と表示されるのはこのとき格納したダウンロード情報中のデータタイプを表示しています。

ダウンロード実行時は、まず指定されたデータがアプリケーションプログラムかどうかを調べるためにこのデータタイプをチェックします。
そして、データタイプがアプリケーションであった場合は、先頭から２５６バイト目からプログラムが格納されているものとして、ここを呼び出します。
冒頭のサブコマンド START において、セクション APP_ID を 0x00800000 番地から配置し、セクション P,C,D を 0x00800100 番地から配置しているのはこの為です。

- P,C,D(00800100)によって、セクション P が 0x00800100 番地から始まり、セクション C、セクション D の順で結合されます。
- セクション Pがプログラム領域です。
- セクション P,C,D については、コンパイラによって自動的に生成されるデフォルトのセクションです。

また、上記の説明から明らかなように、セクション P の先頭には、アプリケーションプログラムのスタートアップコードが格納されている必要があります。
START.ASM 中の__APP_STARTが、アプリケーションプログラムのスタートアップコードとなります。
これをセクション Pの先頭に配置するために、HELLO.SUB の 2 行目

```
INPUT ..\BIOS\start.obj ..\BIOS\init.obj
```

で､最初に start.obj を入力ファイル指定することにより､__APP_START が 0x00800100 番地に配置されます。

③ NP2000-OS とアプリケーションの共有領域(SRAM)の指定

セクション BBIOS は、NP2000-OS とアプリケーションの共有領域となります。
サブコマンド”START BBIOS(00478000)”は、セクション BBIOS の先頭番地を指定しています。
このセクション BBIOS の先頭番地は、絶対に変更しないでください。
NP2000-OS 側も同じセクションを使用し、ここにシステムコール関数へのポインタ等を設定しているためこれを変更すると、アプリケーションプログラムは正常に動作しません。

④ アプリケーションプログラム用 RAM(SRAM メモリー)内のメモリー・マップの指定

アプリケーションプログラム用 RAM 領域(0x00480000-0x004B0000:256KB)については、使用上の制約は特にありません。
本例では、以下のサブコマンドにより RAM 領域のマッピングを行っています。

```
START R,B,BWORK(00480000)
START BBUPMEM(004B0000)
```

セクション R,B は、コンパイラによって生成されるデフォルト・セクションです。
セクション R が初期化データ領域でセクション B が未初期化データ領域です。
INIT.C：APP_INITSCT()関数において、セクション B はすべて 0 にクリアされ、セクション R にはセクション D から初期化データがコピーされます。

NP2000－OS では、動的メモリの管理はサポートしていません。
動的メモリに関しては、セクション BWORK をワークエリアとして使用して、ユーザー自身で管理していただく必要があります。
ワークエリアの管理用に、以下の定数が START.ASM 内で定義されています。

WorkAreaTop	ワークエリア先頭番地
WorkAreaBottom	ワークエリア終了番地
WorkAreaSize	ワークエリアのサイズ

C 言語ソースまたはヘッダーからは以下のように使用してください。

```
extern void *WorkAreaTop;
extern void *WorkAreaBottom;
extern unsigned long WorkAreaSize;
```

NP2000 では、１MB の SRAM 領域はすべてバックアップされています。
但し、バックアップを行いたい変数を通常に宣言すると、セクション R（初期化）、またはセクション B(未初期化)に割り当てられバックアップすることが出来ません。
従って、本例では、セクション BBUPMEM を使用することにより、バックアップメモリとして使用することが出来ます。
セクション BBUPMEM を使用するためには、C 言語ソースまたはヘッダー内で、以下のようにしてセクションを切り替えて、そこにバックアップ変数を宣言します。

```
// バックアップＳＲＡＭ領域用セクションに切り替え
#pragma section BUPMEM
```

```
    // 電池バックアップしたいデータは以下の例のように、ここに記述する。
    int backup_data1;
    int backup_data2;
    int backup_data3;
    int backup_param[100];
// バックアップＳＲＡＭ領域用セクションを終了してデフォルトのセクションに戻す
#pragma section
```

※ C言語ソース上のセクション名はBUPMEMです。BBUPMEMはコンパイラが、BUPMEMの未初期化データ領域として自動的に先頭に’B’を付加するために、リンカに対するサブコマンド指定時はBBUPMEMとなります。

※ バックアップＳＲＡＭ領域につきましては、データの信頼性を確保するために、書き込み時にmk_crc関数を使用してCRCを作成し、適宜err_crc関数によりエラーチェックを実施されることを強く推奨いたします。

8. システムコール

NP2000-OSの機能をアプリケーションプログラムから利用するには、システムコール関数を使用します。システムコール関数を使用するためには、「7. アプリケーションの作成方法」に記したように、ＮＰ2000.hをインクルードする必要があります。
これについては、「7.3 サンプルプログラムのソースファイル作成」および「7.9.1 Ｃ言語ソースプログラムについて」の項をご参照願います。

システムコールだけでは実現できない関数についてはNP2000用ライブラリ(NP2K.LIB)で実装しています。リンク時には必ず、NP2K.LIBをリンクするようにしてください。
本項では、システムコール関数とライブラリ関数を特に区別せず説明しています。

8.1. システム関連システムコール

8.1.1. 【関数名】wait

機能　指定された時間待ちます。
書式　void wait(long t);
引数　long t　　ウェイト時間指定(×ミリ秒)
戻値　なし

■使用例

```
wait(1000);     // １秒間ウェイト
```

8.1.2. 【関数名】GetTickCount

機能　システム起動からの経過時間を取得します。
書式　unsigned long GetTickCount(void);
引数　なし
戻値　システム起動からの経過時間(×ミリ秒)

8.1.3. 【関数名】Buzzer

機能　ブザーを鳴らします。
　　消音は指定時間を過ぎると自動的に行われ当関数はブザーの消音を待ちません。
書式　void Buzzer(int t);
引数　int t　ブザーを鳴らす時間を１０ｍＳ単位で指定します
戻値　なし

8.1.4. 【関数名】mk_crc

機能　指定された領域のＣＲＣを計算し、計算結果を返します。
書式　unsigned short mk_crc(unsigned char *ptr, unsigned long size);
引数　unsigned char *ptr　CRCを計算する領域の先頭番地へのポインター

unsigned long size　CRCを計算する領域のバイトサイズ

戻値　指定された領域のＣＲＣ計算結果。

8.1.5.　【関数名】err_crc

機能　指定された領域のＣＲＣを計算し、与えられたＣＲＣと比較します。

書式　int err_crc(unsigned char *ptr, unsigned long size, unsigned short crc);

引数　unsigned char *ptr　CRCを計算する領域の先頭番地へのポインター

unsigned long size　CRCを計算する領域のバイトサイズ

unsigned short chk　比較する CRC データ

戻値　0　　: OK（ＣＲＣ一致）

0以外: NG（ＣＲＣ一致せず）

8.1.6.　【関数名】RtcRead

機能　時刻・日付の情報を取得します。

書式　int RtcRead(t_rtc *rtc);

引数　t_rtc *rtc　時刻・日付用構造体へのポインター

```
typedef struct {
    unsigned short  year;   西暦年(2000,2001,..)
    unsigned char   mon;    月(1,2,..,12)
    unsigned char   day;    日(1,2,..,31)
    unsigned char   wday;   曜日 0:sunday, 1:monday, .. 6:saturday
    unsigned char   hour;   時(0,1,..,23)
    unsigned char   min;    分(0,1,..,59)
    unsigned char   sec;    秒(0,1,..,59)
} t_rtc;
```

戻値　終了コード 0:読み込み失敗 1:読込み成功

8.1.7.　【関数名】RtcWrite

機能　時刻・日付の情報を設定します。

曜日は本関数内で日付から自動的に計算する為、ydayを設定する必要はありません。

書式　void RtcWrite(t_rtc *rtc);

引数　t_rtc *rtc　時刻・日付用構造体へのポインター

```
typedef struct {
    unsigned short  year;   西暦年(2000,2001,..)
    unsigned char   mon;    月(1,2,..,12)
    unsigned char   day;    日(1,2,..,31)
    unsigned char   wday;   曜日 0:sunday, 1:monday, .. 6:saturday
    unsigned char   hour;   時(0,1,..,23)
```

```
        unsigned char   min;    分(0,1,..,59)
        unsigned char   sec;    秒(0,1,..,59)
    } t_rtc;
```

戻値　なし

8.1.8.　【関数名】LedOut

機能　動作モニター用ＬＥＤを制御します。

書式　void LedOut(int led);

引数　int led　ＬＥＤ出力

　　bit0 : LED0(D10) 0:OFF/1:ON

　　bit1 : LED1(D7) 0:OFF/1:ON

戻値　なし

8.1.9.　【関数名】SysReset

機能　システムにリセットをかけ再起動します。

書式　void SysReset(void);

引数　なし

戻値　なし

8.1.10.　【関数名】BackupDataSave

機能　アプリケーション用に確保された３２KバイトのＦＬＡＳＨメモリー領域に、データを書き込みます。

書式　int BackupDataSave(unsigned char *buf, int len);

引数　unsigned char *buf　書込みデータバッファへのポインター

　　int len　　書込みデータバッファの長さ

戻値　 0 = 正常終了

　　-1 = 書込みに失敗

■注意事項

1.lenが 32Kﾊﾞｲﾄを超えていた場合は、書込みは実行せず書込みに失敗(-1)を返します。

2.len < 0の場合も同様に書込みは実行せず書込みに失敗(-1)を返します。

3.書き込むデータはバックアップすべき全てのデータを書き込んでください。
　データの一部だけの書込みは出来ません。

4.本関数は、ＦＬＡＳＨメモリーの消去、書き換えを行います.
　従って、関数実行中は決して、電源ＯＦＦやリセット等の操作をしないで下さい.

5.本関数は、ＦＬＡＳＨメモリーの消去、書き換えを伴う為、数秒程度時間がかかる場合があります。
　この間、割込み動作は禁止していますので、本関数実行中は、通信やタイマー処理が発

生しないようにしてください。
（通信の取りこぼし、タイマーの誤差等が発生する可能性が有ります）
6.頻繁に本関数を使う事はお勧めできません。
動作が保証されている書込み回数は１０万回です。

8.1.11. 【関数名】BackupDataLoad

機能　アプリケーション用に確保された３２KバイトのＦＬＡＳＨメモリー領域から、データを読み出します。

書式　int BackupDataLoad(unsigned char *buf, int len);

引数　unsigned char *buf　読込みデータバッファへのポインター
　　int len　　読込みデータバッファの長さ

戻値　0 = 正常終了
　　-1 = 読込みに失敗

■注意事項

1.lenが 32Kﾊﾞｲﾄを超えていた場合は、読込み実行せず読込みに失敗(-1)を返します。
2.len < 0の場合も同様に読込みは実行せず読込みに失敗(-1)を返します。

8.1.12. 【関数名】BatChk

機能　バッテリー残量をチェックします。

書式　int BatChk(void)

引数　なし

戻値　1 = ＯＫ
　　0 = バッテリー残量不足（要バッテリー交換）

8.2. ＬＣＤ＆タッチパネル関連システムコール

液晶表示の制御・文字描画・タッチパネル制御を行うシステムコール関数群です。
液晶座標はキャラクタ座標を使用してください。(「4. LCD 表示とタッチパネルの概要」参)

8.2.1. 【関数名】VOLcont

機能 液晶の輝度を設定します。
書式 unsigned char VOLcont(unsigned char vol);
引数 unsigned char vol 1..255: 輝度設定値
0 : 輝度設定は実行せず現在の輝度設定値を返します。
戻値 vol != 0 -> 新しく設定された輝度設定値
vol == 0 -> 現在の輝度設定値

8.2.2. 【関数名】LcdBackLight

機能 バックライトのＯＮ／ＯＦＦを制御します。
書式 void LcdBackLight(int fg);
引数 int fg : 0:OFF/1:ON
戻値 なし

8.2.3. 【関数名】CursorSetup

機能 カーソル位置及びＯＮ／ＯＦＦを制御します。
カーソル座標に(0, 0)を指定した場合は、カーソル位置は移動せず ON/OFF のみを制御します。
書式 void CursorSetup(short x, short y, flag on);
引数 short x,y キャラクタカーソルを表示する座標
flag on カーソル表示の ON(1)/OFF(0)
戻値 なし

8.2.4. 【関数名】LcdPuts

機能 ＬＣＤ画面に文字列を表示します。
表示後のカーソル位置は最終表示位置の次の座標に移動します。
座標に(0, 0)を指定すると現在のカーソル位置より表示します。
書式 void LcdPuts(short x, short y, unsigned char attr, char *str);
引数 short x,y 表示開始キャラクタ座標
unsigned char attr 表示文字のアトリビュート情報（文字単位での設定はできません）
#define LCD_NORMAL 0 ノーマル
#define LCD_BLINK 1 点滅
#define LCD_REVERSE 2 反転

char *str　　表示文字列

戻値　なし

8.2.5.　【関数名】LcdPuts2

機能　液晶画面への文字列表示及び液晶画面の設定を行います。

・アトリビュート情報及びキーテーブル情報を含めた液晶画面の設定を行います。

・アトリビュート情報及びキーテーブル情報は個々に設定可能です。

・カーソル位置は移動しません。

書式　void　LcdPuts2(short x, short y, char　*str,
　　char　*attr1, char　*attr2, char *tpatr);

引数　short x,y　　表示開始キャラクタ座標

char *str　　表示文字列
省略不可のパラメーターです以降のパラメータを設定する場合は
この表示文字列と同じ長さにしてください。
またキーテーブルを設定する場合はX座標は奇数座標から始め長
さは偶数にしてください。

char *attr1　　ブリンクとリバースの情報（NULLを設定すると変更されません）
'A' 'B' 'R' ' ' の４種類の設定が可能です
'B'->ブリンク表示
'R'->反転表示
'A'->反転+ブリンク表示
' '->ノーマル

char *attr2　　罫線情報（NULLを設定すると変更されません）
'0'..'F'の１６進文字で指定します
1->下線
2->上線
4->左線
8->右線
罫線は20*16の文字枠内に引かれます

char *tpatr　　キーテーブル情報（NULLを設定すると変更されません）
２文字の１６進で指定してください

戻値　なし

■使用例１

(1,1)に"確認"を反転表示し、この部分が押されると0x0Dのコードを返します。

```
LcdPuts2(1, 1, "確認", "RRRR", NULL, "0D0D");
```

■使用例２

(3,1)に罫線で囲まれた"確認"を表示し、この部分が押されると0x0Dのコードを返します。
(9,1)に罫線で囲まれた"取消"を表示し、この部分が押されると0x1Bのコードを返します。
LcdPuts2(3, 1, "確認　取消", "　　", "733B00733B, "0D0D001B1B");

8.2.6. 【関数名】ModifyLcdAttr

機能　指定位置のアトリビュート情報を変更します。
　　カーソル位置は移動しません。
書式　void　ModifyLcdAttr(short x, short y, unsigned char attr, short len);
引数　short x,y　　表示開始キャラクタ座標
　　unsigned char attr　表示文字のアトリビュート情報(文字単位での設定はできません)
　　#define LCD_NORMAL　0　ノーマル
　　#define LCD_BLINK　1　点滅
　　#define LCD_REVERSE　2　反転
　　short len　　変更する長さ
戻値　なし

8.2.7. 【関数名】LcdClr

機能　液晶パネル情報の全クリア
　　カーソル位置は（1, 1）に移動し、カーソルＯＦＦになります。
書式　void　LcdClr(void);
引数　なし
戻値　なし

8.2.8. 【関数名】LcdNormal

機能　液晶通常表示
書式　void LcdNormal(void);
引数　なし
戻値　なし

8.2.9. 【関数名】LcdReverse

機能　液晶反転表示
書式　void LcdReverse(void);
引数　なし
戻値　なし

8.2.10. 【関数名】SetupGamen

機能　SHTP.EXEが出力した画面情報を設定します。
　　カーソル位置は移動しません。

書式　void　SetupGamen(short top_x, short top_y, LCD_PANEL　*lpgamen);
引数　short top_x,top_y　　画面情報を設定する左上座標
　　LCD_PANEL lpgamen　SHTP.EXEが出力した画面情報
戻値　なし

※　SHTP.EXEの使用方法につきましては、「SHTP操作説明書」をご覧ください。

8.2.11. 【関数名】isTpHit

機能　タッチパネルの押し下げ状態を検査します。
　　※キーテーブルにコードが設定されている場合にのみ反応します。
　　※無条件にタッチパネルの押し下げを検出するには isTpHit_XY を使用して下さい。
書式　int isTpHit(void);
引数　なし
戻値　０：入力なし　１：入力あり

8.2.12. 【関数名】GetTp

機能　タッチパネル押し下げ情報を取得します。
　　※有効なキー入力が有るまで待ちます。
　　*** 注意 ***
　　キーテーブルに有効な設定が行われていない場合はこの関数はリターンしません。
　　isTpHit関数と組み合わせて用いることをおすすめします。
　　例)　if(isTpHit()) ch = GetTp();
　　※キーテーブルに関係なく無条件にタッチパネル押し下げ位置を検出するには
　　GetTp_XY()を使用します。
書式　unsigned short GetTp(void);
引数　なし
戻値　押下されたキーに対応するキーテーブル上のキー情報

8.2.13. 【関数名】KeyBuzzer

機能　キー入力時のブザーのＯＮ／ＯＦＦを設定します。
書式　void KeyBuzzer(int on);
引数　int on　　キーブザーの設定（０：なし／１：あり）
戻値　なし

8.2.14. 【関数名】isTpHit_XY

機能　無条件にタッチパネルの押し下げ状態を検査します。
書式　int isTpHit_XY(void);
引数　なし

戻値　0 = キー入力なし
　　1 = キー入力あり

8.2.15. 【関数名】GetTp_XY

機能　タッチパネル押し下げ情報を取得します。
　　※入力が有るまで待ちます。
書式　unsigned long GetTp_XY(void);
引数　なし
戻値　押し下げられた位置（Ｘ：０.. １９、Ｙ：０.. １１）
　　上位ワード(D15..8)にX、下位ワード(D7..0)にYを返します。

8.2.16. 【関数名】GetTp_Tbl

機能　タッチパネルのイメージをそのままテーブルとして返します。
　　呼び出し側で、２０×１２＝２４０バイトの領域を確保して渡してください。
　　本関数の戻り値で、キー押下の有無を検出する事も可能ですが、毎回全キーのスキャンを行う為、isTpHit_XY() よりも処理に時間がかかります。
　　isTPHit_XY() と組み合わせて使用することを推奨します。
書式　int GetTp_Tbl(char *tp_tbl, long timeout);
引数　char *tp_tbl　　タッチパネルのイメージ格納領域へのポインター（横２０×縦１２＝２４０バイト）
　　0:キーなし, 1:キー検出
　　複数のキーを押した場合は、押していないキーを検出する事があります。
　　多重キーを許す場合は、キーの配置を十分教慮して下さい。
　　long timeout　　タイムアウト設定（×１０ｍｓｅｃ）
戻値　押下されたキーの有無（0:キーなし, 1:キー有り）

8.2.17. 【関数名】LcdButton

機能　ＬＣＤ画面上にボタンを作成します。
書式　void LcdButton(T_COMPO_BUTTON *btn);
引数　T_COMPO_BUTTON *btn　　ボタンコンポーネント構造体へのポインター
　　typedef struct t_compo_button {
　　struct　t_compo *next;　次のコンポーネントへのポインタ（※１）
　　short　compo;　　コンポーネントの種類（※１）
　　unsigned short left;　左上座標X（テキスト座標：１始まり）
　　unsigned short top;　左上座標Y（テキスト座標：１始まり）
　　unsigned short width;　表示幅
　　unsigned short height;　表示高
　　unsigned char border;　ボタン枠の指定(1:単線 2:２重線)（※２）

```
    unsigned char attr;     文字の属性
    char *caption;          キャプション（※３）
    unsigned char layout;   配置（※１）
    char key_code;          ボタンに割り付けるキーコード
} T_COMPO_BUTTON;
```

戻値　なし

■注意事項

※１　これらの項目は現在使用していません。NULL または 0 を指定して下さい。

※２　２重線を使用する場合は、表示高は枠線用に１行余分に必要です。
例えば、サンプルプログラム HELLO で作成した、右下の「サンプル終了」ボタンは、キャプション（表題）は、下図のように２行ですが、ボタン全体の高さは３行分使用しています。

また、表示幅に関しては、単線、２重線共に、枠専用に全角１文字分余分に必要です。

※３　キャプション（表題）は”\n”を使用することにより改行することが出来ます。
例えば、上の例ではキャプション文字列を”サンプル\n終　　了”とする事により２行に表示しています。

■使用例１

サンプルプログラム HELLO での「サンプル終了」ボタンでの使用例です。（※２写真参）

```
T_COMPO_BUTTON exit_btn  = {NULL, 0, 31, 10, 10, 3, 2, LCD_REVERSE, "サンプル\n終　　了", 0, 0x1b};
LcdButton(&exit_btn);
```

- ※２、※３で記したように、キャプションは２行表示ですが、枠線のスペースを確保するため、表示高は３にしてあります。
- ※２で記したように、キャプションの文字幅は全角４文字ですが、枠線のスペースを確保するため、表示幅は１０にしてあります。

■使用例２

使用例１のボタン枠を単線にした例です。

```
T_COMPO_BUTTON exit_btn  = {NULL, 0, 31, 10, 10, 2, 1, LCD_REVERSE, "サンプル\n終　　了", 0, 0x1b};
LcdButton(&exit_btn);
```

- 枠線のスペースを確保する必要が無いため、表示高は２にしてあります。
- 表示幅に関しては、使用例１と同様に１０にしてあります。

8.2.18. 【関数名】LcdPrintf

機能　書式付きでＬＣＤに文字を表示します

書式　int LcdPrintf(short x, short y, unsigned char attr, char *fmt, ...);

引数　short x,y　　表示開始キャラクタ座標
　　座標に(0, 0)を指定すると現在のカーソル位置より表示します。
　　表示後のカーソル位置は最終表示位置の次の座標です。

　　unsigned char attr　表示文字のアトリビュート情報（文字単位での設定はできません）
　　#define LCD_NORMAL　0　ノーマル
　　#define LCD_BLINK　1　点滅
　　#define LCD_REVERSE　2　反転

　　char *fmt　　書式文字列（※１、※２、※３）
　　改行・復帰・タブ等のエスケープ文字は使えません

戻値　正常終了：出力文字数（≧０）
　　異常終了：エラーコード（＜０）

■使用例

```
int val = 777;
LcdPrintf(10, 10, LCD_BLINK, "最大値=%d", val);
```

■注意事項

※１　フィールド幅あるいは精度に対する *指定はサポートしておりません。

※２　変換文字 n(変換文字数の格納)はサポートしておりません。

※３　書式指定の詳細については、日立 C/C++コンパイラ・マニュアルの「標準ライブラリ－fprintf 関数」の項をご参照願います。

8.2.19. 【関数名】GetTpTable

機能　現在設定されているキーコードテーブルを読み出します。

書式　void GetTpTable(unsigned char *tp_tbl);
引数　unsigned char *tp_tbl　キーコードテーブルの格納領域へのポインター
本関数は、キーコードテーブルを無条件に tp_tbl コピーします。
従って、呼出側でタッチパネルのキー数（２０×１２）分の配列を確保し、その先頭アドレスを渡してください。
戻値　なし

8.2.20. 【関数名】SetTpTable

機能　タッチパネルのキーコード・テーブルを直接設定します。
書式　void SetTpTable(unsigned char *tp_tbl);
引数　unsigned char *tp_tbl　設定するキーコードテーブルへのポインター
必ず呼び出し側でタッチパネルのキー数（２０×１２）分の配列を確保しそこにキーコードテーブルを設定し、その先頭アドレスを渡してください。
戻値　なし

8.3. ＲＳ－ＣＯＭＭ関連システムコール

シリアル通信制御を行うシステムコール関数群です。

COM1はRS232Cポート、COM2はRS485が割り当てられています。

RS485ポートは、Rs485Init関数により、全２重、半２重をプログラム上から設定できます。

従って、RS485 ポート使用時は、RsCommOpen関数による COM ポートの初期化に続いて必ず Rs485Init 関数によってRS485トランシーバの初期化を行ってください。

printd関数、ercd_print関数、DebugPrintf関数は開発・メンテナンス用ポートに情報を出力するための関数です。

8.3.1. 【関数名】RsCommOpen

機能　ＣＯＭポートの初期化を行います。

※既に初期化されたポートの再初期化はサポートしていません。
RsCommClose() でクローズ後、オープンしなおしてください。

書式　void RsCommOpen(int com_num, const char *param, int *ercd);

引数　int com_num　COMポート番号(1,2,3,4)
1:RS232C
2:RS485
3:拡張ＲＳ２３２Ｃ(COM3)
4:拡張ＲＳ２３２Ｃ(COM4)

char *param　パラメータ文字列

ﾎﾞｰﾚｰﾄ	"9600" etc.(“57600”MAX)
データ長	"B8"=8bit "B7"=7bit
ｽﾄｯﾌﾟBIT	"S1"=1bit "S2"=2bit
ﾊﾟﾘﾃｨｰ	"PN"=none "PE"=even "PO"=odd
ﾌﾛｰ制御	指定せず=ﾌﾛｰ制御なし
	"XON"=XON/OFFによるフロー制御有り。
	"RTS"=RTSによるフロー制御有り。
無指定時	"9600 B8 PN S1"

int *ercd　エラーコード格納先へのポインター
ＮＵＬＬ設定時は、エラーコードを返しません。

戻値　なし

■ 使用例

```
int ercd;
RsCommOpen( 1, "9600 B8 PN S1", &ercd);
```

8.3.2. 【関数名】RsCommClose

機能 指定されたＣＯＭポートをクローズします

書式 void RsCommClose(void);

引数 int com_num COMポート番号(1,2,3,4)
1:RS232C
2:RS485
3:拡張ＲＳ２３２Ｃ(COM3)
4:拡張ＲＳ２３２Ｃ(COM4)

戻値 なし

8.3.3. 【関数名】RsCommCtrl

機能 ＣＯＭポートを制御します。

書式 void RsCommCtrl(int com_num, unsigned short fncd, int *ercd);

引数 int com_num COMポート番号(1,2,3,4)
1:RS232C
2:RS485
3:拡張ＲＳ２３２Ｃ(COM3)
4:拡張ＲＳ２３２Ｃ(COM4)

unsigned short fncd 機能コード
機能コードは以下の定数を単独、或いは組み合わせて指定します。

FNCD_COMM_RXE	：受信イネーブル
FNCD_COMM_RXD	：受信ディセーブル
FNCD_COMM_TXE	：送信イネーブル
FNCD_COMM_TXD	：送信ディセーブル
FNCD_COMM_RTSON	：RTS信号ON
FNCD_COMM_RTSOFF	：RTS信号OFF
FNCD_COMM_DTRON	：DTR信号ON
FNCD_COMM_DTROFF	：DTR信号OFF
FNCD_COMM_RXCLR	：受信バッファクリア
FNCD_COMM_TXCLR	：送信バッファクリア
FNCD_COMM_SBON	：ブレーク送信ON
FNCD_COMM_SBOFF	：ブレーク送信OFF

int *ercd エラーコード格納先へのポインター
ＮＵＬＬ設定時は、エラーコードを返しません。

戻値 なし

8.3.4. 【関数名】RsCommStat

機能 ＣＯＭポートの状態を取得します。

書式 void RsCommStat(int com_num, T_COMMSTAT *stat, int *ercd);

引数 int com_num　　COMポート番号(1,2,3,4)
　　1:RS232C
　　2:RS485
　　3:拡張ＲＳ２３２Ｃ(COM3)
　　4:拡張ＲＳ２３２Ｃ(COM4)

T_COMMSTAT *stat　　ＣＯＭポート入出力ステータス構造体へのポインター

```
typedef struct t_commstat
{
    unsigned short stat;        シリアル入出力ステータス
      ステータスは以下の定数との論理積により抽出できます
      STAT_COMM_CD    :  受信キャリア検出
      STAT_COMM_CTS   :  CTS信号 ON(1)/OFF(0)
      STAT_COMM_TXEMP :  送信バッファ空
      STAT_COMM_PE    :  パリティエラー
      STAT_COMM_OE    :  オーバランエラー
      STAT_COMM_FE    :  フレーミングエラー
      STAT_COMM_BD    :  ブレーク状態検出
      STAT_COMM_DSR   :  DSR信号 ON(1)/OFF(0)
      STAT_COMM_RI    :  RI検出
    unsigned char rxchr;        受信バッファの先頭の文字（未使用）
    unsigned short rxlen;       受信バッファのデータ長（受信バイト数）
    unsigned short txlen;       送信バッファのデータ長（送信待ちバイト数）
    unsigned short frbufsz;     送信バッファの空きサイズ
    unsigned short eotcnt;      受信バッファの終端文字個数
} T_COMMSTAT;
```

int *ercd　　エラーコード格納先へのポインター
　　ＮＵＬＬ設定時は、エラーコードを返しません。

戻値 なし

8.3.5. 【関数名】RsCommGetch

機能 ＣＯＭポート受信バッファから、１文字取得します。

書式 int RsCommGetch(int com_num, long timeout, int *ercd);

引数 int com_num　　COMポート番号(1,2,3,4)
　　1:RS232C
　　2:RS485

3:拡張ＲＳ２３２Ｃ(COM3)
4:拡張ＲＳ２３２Ｃ(COM4)
long timeout　タイムアウト設定（×１０ｍｓｅｃ）
int *ercd　エラーコード格納先へのポインター
ＮＵＬＬ設定時は、エラーコードを返しません。
戻値　受信文字

【解説】

ＣＯＭポートから受信バッファへの受信処理は、受信割り込みが行います。
本システムコールでは、受信バッファから１文字を取得します。
受信バッファが空であった場合は、データを受信するまで待ち状態になります。
タイムアウト無し(timeout=-1)で本システムコールを呼び出した場合、受信が完了するまで待ち状態が続きます。
タイムアウト有り(timeout=1～0x7ffffff)で本システムコールを呼び出した場合は、指定した時間が経過してもデータを受信しなければ本システムコールは処理を中止し*ercd に E_TMOUT を格納します。
タイムアウト無効（timeout=0）で本システムコールを呼び出すと、受信バッファが空であった場合は直ぐに処理を中止し*ercd に E_TMOUT を格納します。

8.3.6. 【関数名】RsCommPutch

機能　ＣＯＭポート送信バッファに１文字格納します。
書式　void RsCommPutch(int com_num, char c, long timeout, int *ercd);
引数　int com_num　COM ポート番号(1,2,3,4)
1:RS232C
2:RS485
3:拡張ＲＳ２３２Ｃ(COM3)
4:拡張ＲＳ２３２Ｃ(COM4)
char c　送信文字
long timeout　タイムアウト設定（×１０ｍｓｅｃ）
int *ercd　エラーコード格納先へのポインター
ＮＵＬＬ設定時は、エラーコードを返しません。
戻値　なし

【解説】

送信バッファからＣＯＭポートへの送信処理は、送信割り込みが行います。
本システムコールは、送信バッファへ１文字格納します。
送信バッファが一杯であった場合は、送信バッファに空きが生じるまで待ち状態になります。
タイムアウト無し(timeout=-1)で本システムコールを呼び出した場合、送信バッファに空きが生じ

るまで待ち状態が続きます。
タイムアウト有り(timeout=1～0x7ffffff)で本システムコールを呼び出した場合、指定した時間が経過しても送信バッファに空きが生じなければ本システムコールは処理を中止し*ercd に E_TMOUT を格納します。
タイムアウト無効（timeout=0）で本システムコールを呼び出すと、送信バッファに空きが無ければ直ぐに処理を中止し*ercd に E_TMOUT を格納します。

8.3.7. 【関数名】RsCommKbHit

機能　ＣＯＭポート受信バッファに受信データがあるかどうかを調べます。

書式　int RsCommKbHit(int com_num, int *ercd);

引数　int com_num　　COM ポート番号(1,2,3,4)
　　1:RS232C
　　2:RS485
　　3:拡張ＲＳ２３２Ｃ(COM3)
　　4:拡張ＲＳ２３２Ｃ(COM4)
　　int *ercd　　エラーコード格納先へのポインター
　　ＮＵＬＬ設定時は、エラーコードを返しません。

戻値　0　　受信データなし
　　0 以外　受信データ有り（受信バイト数）

8.3.8. 【関数名】RsCommGets

機能　ＣＯＭポート受信バッファから文字列を取得します。
　　改行文字を読み込むか、len-1 個の文字を読み込んだ場合に終了します。

書式　char *RsCommGets(int com_num, char *buf, int len, int *bytes, char echo,
　　long timeout, int *ercd);

引数　int com_num　　COM ポート番号(1,2,3,4)
　　1:RS232C
　　2:RS485
　　3:拡張ＲＳ２３２Ｃ(COM3)
　　4:拡張ＲＳ２３２Ｃ(COM4)
　　char *buf　　読み込んだ文字列を格納するバッファ
　　int len　　バッファの長さ
　　int *bytes　　実際に読み込んだ文字数の格納先へのポインター
　　ＮＵＬＬ設定時は、文字数は返しません。
　　char echo　　エコー設定
　　0　　:エコーバックしません。
　　'*'　　:'*'をエコーバックします
　　上記以外　:エコーバックをします。

long timeout　タイムアウト設定（×１０ｍｓｅｃ）
int *ercd　エラーコード格納先へのポインター
ＮＵＬＬ設定時は、エラーコードを返しません。

戻値　文字列に対するポインター（エラー時はＮＵＬＬ）

【解説】

ＣＯＭポートから受信バッファへの受信処理は、受信割り込みが行います。
本システムコールは、受信バッファから文字列の取得を行います。
文字列取得中に受信バッファが空になった場合は、データを受信するまで待ち状態になります。
また、エコーバック送信中に送信バッファが一杯になった場合は、送信バッファに空きが生じるまで待ち状態になります。
タイムアウトは文字列取得動作全体ではなく、１文字受信および１文字送信（エコーバック）の動作に対して設定されます。
タイムアウト無し(timeout=-1)で本システムコールを呼び出した場合、１文字受信が完了するまで待ち状態が続きます。
タイムアウト有り(timeout=1～0x7ffffff)で本システムコールを呼び出した場合は、指定した時間が経過してもデータを受信しなければ本システムコールは処理を中止し、*bytesにそれまでに取得できた文字数、*ercdにE_TMOUTを格納し、NULLを返します。
タイムアウト無効（timeout=0）で本システムコールを呼び出すと、受信バッファが空であった場合は直ぐに処理を中止し、*bytesにそれまでに取得できた文字数、*ercdにE_TMOUTを格納し、NULLを返します。

エコーバック時も同様に、タイムアウト無し(timeout=-1)で本システムコールを呼び出した場合、送信バッファに空きが生じるまで待ち状態が続きます。
タイムアウト有り(timeout=1～0x7ffffff)で本システムコールを呼び出した場合、指定した時間が経過しても送信バッファに空きが生じなければ本システムコールは処理を中止し、*bytesにそれまでに取得できた文字数、*ercdにE_TMOUTを格納し、NULLを返します。
タイムアウト無効（timeout=0）で本システムコールを呼び出すと、送信バッファに空きが無ければ直ぐに処理を中止し、*bytesにそれまでに取得できた文字数、*ercdにE_TMOUTを格納し、NULLを返します。

■注意事項

RS485ポート(COM2)の場合、RsCommGets() 時のエコーバック出力に関しても、自動制御が有効です。
従って、複数の機器がＲＳ４８５回線に接続されている状態では、自動制御をしない、エコーバックをしない、あるいはRsCommGets()より前で予め調停を行い、選択されているユニットのみがRsCommGets()を行う、等の対策を講じて、エコーバックによるデータの衝突が発生しないようにして下さい。

8.3.9. 【関数名】RsCommPuts

機能　ＣＯＭポート送信バッファへ文字列を格納します。

書式　void RsCommPuts(int com_num, const char *str, int *bytes,
long timeout, int *ercd);

引数	int com_num	COMポート番号(1,2,3,4) 1:RS232C 2:RS485 3:拡張ＲＳ２３２Ｃ(COM3) 4:拡張ＲＳ２３２Ｃ(COM4)
	const char *str	送信文字列
	int *bytes	実際に格納できた文字数の格納先へのポインター ＮＵＬＬ設定時は、文字数は返しません。
	long timeout	タイムアウト設定（×１０ｍｓｅｃ）
	int *ercd	エラーコード格納先へのポインター ＮＵＬＬ設定時は、エラーコードを返しません。

戻値　なし

【解説】

送信バッファからＣＯＭポートへの送信処理は、送信割り込みが行います。
本システムコールは、送信バッファに指定された文字列を格納します。
文字列格納中に送信バッファが一杯になった場合は、送信バッファに空きが生じるまで待ち状態になります。
タイムアウトは文字列格納動作全体ではなく、１文字格納動作に対して設定されます。
タイムアウト無し(timeout=-1)で本システムコールを呼び出した場合、送信バッファに空きが生じるまで待ち状態が続きます。
タイムアウト有り(timeout=1～0x7ffffff)で本システムコールを呼び出した場合、指定した時間が経過しても送信バッファに空きが生じなければ本システムコールは処理を中止し、*bytes にそれまでに格納できた文字数、*ercd に E_TMOUT を格納します。
タイムアウト無効（timeout=0）で本システムコールを呼び出すと、送信バッファに空きが無ければ直ぐに処理を中止し、*bytes にそれまでに格納できた文字数、*ercd に E_TMOUT を格納します。

8.3.10. 【関数名】RsCommRead

機能　ＣＯＭポート受信バッファからデータを指定バイト数だけ読み込みます。

書式　int RsCommRead(int com_num, unsigned char *buf, int len,
long timeout, int *ercd);

引数	int com_num	COMポート番号(1,2,3,4) 1:RS232C 2:RS485

3:拡張ＲＳ２３２Ｃ(COM3)
4:拡張ＲＳ２３２Ｃ(COM4)

unsigned char *buf	読み込んだデータを格納するバッファへのポインター
int len	要求受信長
long timeout	タイムアウト設定（×１０ｍｓｅｃ）
int *ercd	エラーコード格納先へのポインター ＮＵＬＬ設定時は、エラーコードを返しません。

戻値　実際に読み込んだバイト数

【解説】

ＣＯＭポートから受信バッファへの受信処理は、受信割り込みが行います。
本システムコールは、受信バッファから指定バイト数のデータ読込み、指定されたバッファへ格納します。
本システムコールの処理中に受信バッファが空になった場合は、データを受信するまで待ち状態になります。
タイムアウトは本システムコールの動作全体ではなく、１バイト受信の動作に対して設定されます。
タイムアウト無し(timeout=-1)で本システムコールを呼び出した場合、１バイト受信が完了するまで待ち状態が続きます。
タイムアウト有り(timeout=1～0x7ffffff)で本システムコールを呼び出した場合は、指定した時間が経過してもデータを受信しなければ本システムコールは処理を中止し、*ercd に E_TMOUT を格納し、それまでに読み込んだバイト数を返します。
タイムアウト無効（timeout=0）で本システムコールを呼び出すと、受信バッファが空であった場合は直ぐに処理を中止し、*ercd に E_TMOUT を格納し、それまでに読み込んだバイト数を返します。

8.3.11. 【関数名】RsCommWrite

機能　ＣＯＭポート送信バッファへデータを指定バイト数だけ格納します。

書式　int RsCommWrite(int com_num, unsigned char *buf, int len,
long timeout, int *ercd);

引数　int com_num	COM ポート番号(1,2,3,4) 1:RS232C 2:RS485 3:拡張ＲＳ２３２Ｃ(COM3) 4:拡張ＲＳ２３２Ｃ(COM4)
unsigned char *buf	送信データ格納バッファへのポインター
int len	要求送信長
long timeout	タイムアウト設定（×１０ｍｓｅｃ）
int *ercd	エラーコード格納先へのポインター ＮＵＬＬ設定時は、エラーコードを返しません。

戻値　実際に格納出来たバイト数

【解説】

送信バッファからＣＯＭポートへの送信処理は、送信割り込みが行います。
本システムコールは、指定されたバッファから指定バイト数を読み込み、送信バッファへ格納します。
本システムコール処理中に送信バッファが一杯になった場合は、送信バッファに空きが生じるまで待ち状態になります。
タイムアウトは本システムコールの動作全体ではなく、１バイト送信の動作に対して設定されます。
タイムアウト無し(timeout=-1)で本システムコールを呼び出した場合、送信バッファに空きが生じるまで待ち状態が続きます。
タイムアウト有り(timeout=1～0x7ffffff)で本システムコールを呼び出した場合、指定した時間が経過しても送信バッファに空きが生じなければ本システムコールは処理を中止し、*ercd に E_TMOUT を格納し、それまでに格納できたバイト数を返します。
タイムアウト無効（timeout=0）で本システムコールを呼び出すと、送信バッファに空きが無ければ直ぐに処理を中止し、*ercd に E_TMOUT を格納し、それまでに格納できたバイト数を返します。

8.3.12. 【関数名】RsCommPrintf

機能　書式付でＣＯＭポートの送信バッファへ文字列を送信します。

書式　int RsCommPrintf(int com_num, char *fmt, ...);

引数　int com_num　　COM ポート番号(1,2,3,4)
　　1:RS232C
　　2:RS485
　　3:拡張ＲＳ２３２Ｃ(COM3)
　　4:拡張ＲＳ２３２Ｃ(COM4)
　　char *fmt　　書式文字列（※１、※２、※３）

戻値　正常終了：出力文字数（≧０）
　　異常終了：エラーコード（＜０）

【解説】

書式変換後の文字列の送信は、RsCommPuts システムコールを使用します。
RsCommPuts を呼び出す場合の、タイムアウト設定はタイムアウト無し(-1)で行います。
文字列送信動作の解説は RsCommPuts システムコールをご参照願います。

■注意事項

※１　フィールド幅あるいは精度に対する *指定はサポートしておりません。

※２　変換文字 n(変換文字数の格納)はサポートしておりません。

※３　書式指定の詳細については、日立C/C++コンパイラ・マニュアルの「標準ライブラリ－fprintf関数」の項をご参照願います。

8.3.13. 【関数名】Rs485Init

機能　ＲＳ４８５トランシーバーを初期化します。
　　ドライバはＯＦＦ、レシーバはＯＮに設定されます。

書式　void Rs485Init(int half, int txrv, int rxrv, int auto_drv);

引数　int half　　全２重、半２重の設定 0:全２重, 1:半２重
　　int txrv　　ドライバーの極性を反転します。 0:通常, 1:反転
　　int rxrv　　レシーバの極性を反転します。　0:通常, 1:反転
　　int auto_drv　0 : 送信時のドライバ・レシーバの制御を自動で行いません。
　　Rs485RcvCtrl(), Rs485DrvCtrl() を使用して、手動で制御して下さい。
　　1 : 送信時のドライバ・レシーバの制御を自動で制御します。
　　[全２重時]
　　送信時のみドライバをＯＮにし、送信終了時にＯＦＦにします。
　　レシーバの制御は行いません。
　　[半２重時]
　　送信時のみドライバをＯＮにし、送信終了時にＯＦＦにします。
　　送信時はレシーバをＯＦＦにし、送信終了時にＯＮにします。

戻値　なし

■注意事項

RsCommGets() 時のエコーバック出力に関しても、自動制御が有効です。
従って複数の機器がＲＳ４８５回線に接続されている状態では、自動制御をしない、エコーバックをしない、あるいはRsCommGets()より前で予め調停を行い、選択されているユニットのみがRsCommGets()を行う、等の対策を講じて、エコーバックによるデータの衝突が発生しないようにして下さい。

8.3.14. 【関数名】Rs485RcvCtrl

機能　ＲＳ４８５レシーバーの制御を行います。
　　半２重回線では、レシーバをＯＮのまま、送信を行うと送信データがループバックされます。
　　ループバックをさせない為には、送信時はレシーバをＯＦＦにして下さい。

書式　void Rs485RcvCtrl(int ena);

引数　int ena　　0:レシーバーをＯＦＦし受信禁止, 1:レシーバーをＯＮし受信許可

戻値　なし

■補足事項

データの受信には RsCommGetch(), RsCommGets(), RsCommRead() 等を使用して下さい。

8.3.15. 【関数名】Rs485DrvCtrl

機能　ＲＳ４８５ドライバの制御を行います。

書式　void Rs485DrvCtrl(int ena);

引数　int ena　　0:ドライバをＯＦＦし送信禁止, 1:ドライバをＯＮし送信許可

戻値　なし

■補足事項

データの送信には RsCommPuch(), RsCommPuts(), RsCommWrite() 等を使用して下さい。

8.3.16. 【関数名】printd

機能　デバッグポートに文字列を出力します。

書式　void printd(char *str);

引数　出力文字列

戻値　なし

8.3.17. 【関数名】ercd_print

機能　指定されたエラーコードの値及びエラーの内容をデバッグポートに出力します。

　　※エラーコードが E_OK（正常終了）の場合は何も表示しません。

書式　void ercd_print(const char *cmnt, int ercd);

引数　int *cmnt　エラーの説明文字列（先頭に付加されます）

　　int ercd　　エラーコード

戻値　なし

■補足事項

表示結果は、ＯＳ内部のエラーログ記録領域（最大１６Ｋバイト）に保存されます。

エラーの記録は errchk コマンドで表示できます。

エラーの記録は errclr コマンドでクリアできます。

但し、以下の場合はエラーを記録しません。

①エラーログ記録領域にＣＲＣエラーが発生している場合。

②エラーログのサイズが記録領域の最大サイズを超えてしまう場合。

8.3.18. 【関数名】DebugPrintf

機能　書式付でデバッグポートに文字列を出力します。

書式　int DebugPrintf(char *fmt, ...);

引数　char *fmt　　書式文字列（※１、※２、※３）

戻値　正常終了：出力文字数（≧０）

　　異常終了：エラーコード（＜０）

■注意事項

※１　フィールド幅あるいは精度に対する *指定はサポートしておりません。

※２　変換文字 n(変換文字数の格納)はサポートしておりません。

※３　書式指定の詳細については、日立 C/C++コンパイラ・マニュアルの「標準ライブラリ－fprintf 関数」の項をご参照願います。

8.4. ＴＣＰ関連システムコール

ＴＣＰ／ＩＰ通信を行うためのシステム関数群です。

ＴＣＰ通信は NP2000-OS 独自のソケットを使用して行います。

ソケットには以下の 2 種類があります。

・能動オープン用ソケット

NP2000 側から、接続先に接続要求を発行して接続を確立させるタイプのソケットです。

接続要求の発行は TcpConnect 関数を使用します。

・受動オープン用ソケット

NP2000 側から、接続要求を発行するのではなく、相手先からの接続要求を待ち受けて、接続要求があった場合に通信を確立させるタイプのソケットです。

接続要求待ちは TcpAccept 関数を使用します。

8.4.1. 【関数名】TcpSocket

機能　新規ソケットを生成します。

書式　SOCKET　TcpSocket(int socket_type, unsigned long ip_addr,
unsigned short portno);

引数　int socket_type　　ソケットの種類を以下の定数で指定します。
ACTIVE_SOCKET　能動オープン用ソケットを生成します。
PASSIVE_SOCKET　受動オープン用ソケットを生成します。

unsigned long ip_addr　受動オープン時の接続待ちＩＰアドレス
０を指定すると default_ipaddr を使用します。
能動オープン用ソケットでは使用しません。

unsigned short portno　受動オープン時の接続待ちポート番号
能動オープン用ソケットでは使用しません。

戻値　０以上　ソケット・ディスクリプタ

負の値　エラーコード
E_PAR=空きソケットが無いか、ソケットの種類が不正です。
その他標準エラーコード

8.4.2. 【関数名】TcpAccept

機能　受動オープン用ソケットの接続要求待ちを行います。

書式　int TcpAccept(SOCKET s, int nonblk);

引数　SOCKET s　　ソケット・ディスクリプタ

int nonblk　　終了モードを指定します。
0 = 接続が完了するか、エラーが発生するまで待ちます。(Blocking Call)
1 = 接続の完了を待たずに終了します。(NonBlocking Call)

この場合、接続の完了は TcpIsConnect() 関数で調べます。

戻値　エラーコード

E_NOEXS = ソケット未生成

E_PAR　= ソケット・ディスクリプタの値が範囲外である.
指定されたソケットは能動オープン用ソケットである。

その他標準エラーコード

8.4.3.　【関数名】TcpConnect

機能　能動オープン用ソケットを使用して接続要求を出します。

書式　int TcpConnect(SOCKET s, unsigned long ip_addr, unsigned short portno,
int nonblk);

引数　SOCKET s　ソケット・ディスクリプタ

unsigned long ip_addr　接続先のＩＰアドレス

unsigned short portno　接続先のポート番号

int nonblk　終了モードを指定します。

0 = 接続が完了するか、エラーが発生するまで待ちます。(Blocking Call)

1 = 接続の完了を待たずに終了します。(NonBlocking Call)
この場合、接続の完了は TcpIsConnect() 関数で調べます。

戻値　エラーコード

E_NOEXS = ソケット未生成

E_PAR　= ソケット・ディスクリプタの値が範囲外である.
指定されたソケットは受動オープン用ソケットである。

その他標準エラーコード

8.4.4.　【関数名】TcpIsConnect

機能　指定されたソケットの接続状況を調べます。

書式　int TcpIsConnect(SOCKET s);

引数　SOCKET s　ソケット・ディスクリプタ

戻値　接続状況

CS_OK　= 接続完了

E_PAR　= ソケット・ディスクリプタの値が範囲外である。

E_NOEXS　= ソケット未生成

CS_CRE_SOCK = ソケット生成後コネクト処理(TcpAccept/TcpConnect)待ち。
TcpClose()による正常切断時もこのステータスが返ります。

CS_CON_WAI　= 能動オープン(NonBlockingCall)による接続待ち。

CS_ACP_WAI　= 受動オープン(NonBlockingCall)による接続要求待ち。

CS_DIS_WAI　= 切断処理（NonBlockingCall）による切断待ち。

CS_RCV_FIN = TCP 接続が正常に切断された(FIN 受信)。
CS_RCV_RST = TCP 接続が強制的に切断された(RST 受信)。
CS_SOCK_ERR = TCP 接続に異常が発生しました。
その他標準エラーコード

8.4.5. 【関数名】TcpClose

機能 指定されたソケットを切断します。
ソケット自身のクローズは行わないため、TcpAccept()/TcpConnect() によって再度接続できます。

書式 int TcpClose(SOCKET s, int nonblk, long timeout);

引数 SOCKET s ソケット・ディスクリプタ
int nonblk 終了モードを指定します。
0 = 切断が完了するか、エラーが発生するまで待ちます。
1 = 切断の完了を待たずに終了します。
この場合、切断の完了は TcpIsConnect() 関数で調べます。
long timeout ブロッキングモード時のタイムアウト

戻値 エラーコード
E_PAR = ソケット・ディスクリプタの値が範囲外である.
E_NOEXS = ソケット未生成
E_OBJ = ソケット・ステータス異常
E_SOCK_CLOSE = ソケット接続の切断に失敗、再起動が必要です
その他標準エラーコード

8.4.6. 【関数名】TcpSocketClose

機能 指定されたソケットを切断し、ソケットをクローズします。
切断のみを実行したい場合は TcpClose() を使用してください。

書式 int TcpSocketClose(SOCKET s , int nonblk, long timeout);

引数 SOCKET s ソケット・ディスクリプタ
int nonblk 終了モードを指定します。
0 = 切断が完了するか、エラーが発生するまで待ちます。
1 = 切断の完了を待たずに終了します。
この場合、切断の完了は TcpIsConnect() 関数で調べます。
long timeout ブロッキングモード時のタイムアウト

戻値 エラーコード
E_OK = ソケットのクローズに成功
E_SOCK_CLOSE = ソケットのクローズに失敗、再起動が必要です

8.4.7. 【関数名】TcpSocketCloseAll

機能　全てのソケットをクローズします。

書式　int TcpSocketCloseAll(int nonblk, long timeout);

引数　int nonblk　終了モードを指定します。

0 = 切断が完了するか、エラーが発生するまで待ちます。

1 = 切断の完了を待たずに終了します。

この場合、切断の完了は TcpIsConnect() 関数で調べます。

long timeout　ブロッキングモード時のタイムアウト

戻値　エラーコード

8.4.8. 【関数名】TcpRead

機能　指定バイト数だけＴＣＰ受信バッファからデータを読み込みます。

書式　int TcpRead(SOCKET s, unsigned char *buf, int len, long timeout,

int *ercd);

引数	SOCKET s	ソケット・ディスクリプタ
	unsigned char *buf	受信データ格納エリアへのポインター
	int len	要求受信長
	long timeout	タイムアウト設定（×１０ｍｓｅｃ）
	int *ercd	エラーコード格納先へのポインター ＮＵＬＬ設定時は、エラーコードを返しません。

戻値　実際に読み込んだバイト数

【解説】

ＴＣＰ／ＩＰポートから受信バッファへの受信処理は、ＴＣＰ／ＩＰ受信タスクが行います。

本システムコールは、受信バッファから指定バイト数のデータを読み込み、指定されたバッファへ格納します。

本システムコールの処理中に受信バッファが空になった場合は、ＴＣＰ／ＩＰ受信タスクに受信要求を発行し、受信バッファにデータが格納されるまで待ち状態になります。

タイムアウトは本システムコールの動作全体ではなく、ＴＣＰ／ＩＰ受信タスクへの受信要求の発行に対して設定されます。

タイムアウト無し(timeout=-1)で本システムコールを呼び出した場合、受信が発生するまで待ち状態が続きます。

タイムアウト有り(timeout=1～0x7ffffff)で本システムコールを呼び出した場合は、指定した時間が経過してもデータを受信しなければ本システムコールは処理を中止し、*ercd に E_TMOUT を格納し、それまでに読み込んだバイト数を返します。

タイムアウト無効（timeout=0）で本システムコールを呼び出すと、受信バッファが空であった場合は直ぐに処理を中止し、*ercd に E_TMOUT を格納し、それまでに読み込んだバイト数を返します。

8.4.9. 【関数名】TcpWrite

機能　指定バイト数だけＴＣＰ送信バッファにデータを格納します。
TcpWrite()では、ＴＣＰ送信バッファに空きがある場合は、ＴＣＰ送信バッファに書き込むだけで送信処理は行いません。
ＴＣＰ送信バッファ中のデータを、強制的に送信する場合は、TcpPush()を使用して下さい。

書式　int TcpWrite(SOCKET s, unsigned char *buf, int len, long timeout,
int *ercd);

引数		
	SOCKET s	ソケット・ディスクリプタ
	unsigned char *buf	送信データ格納エリアへのポインター
	int len	要求送信長
	long timeout	タイムアウト設定（×１０ｍｓｅｃ）
	int *ercd	エラーコード格納先へのポインター ＮＵＬＬ設定時は、エラーコードを返しません。

戻値　実際に格納出来たバイト数

【解説】

送信バッファからＴＣＰ／ＩＰポートへの送信処理は、ＴＣＰ／ＩＰ送信タスクが行います。
本システムコールは、指定されたバッファから指定バイト数を読み込み、送信バッファへ格納します。
本システムコール処理中に送信バッファが一杯になった場合は、ＴＣＰ／ＩＰ送信タスクに送信要求を発行し、送信バッファに空きが生じるまで待ち状態になります。
タイムアウトは本システムコールの動作全体ではなく、送信バッファの空き待ちに対して設定されます。
タイムアウト無し(timeout=-1)で本システムコールを呼び出した場合、送信バッファに空きが生じるまで待ち状態が続きます。
タイムアウト有り(timeout=1～0x7ffffff)で本システムコールを呼び出した場合、指定した時間が経過しても送信バッファに空きが生じなければ本システムコールは処理を中止し、*ercd に E_TMOUT を格納し、それまでに格納できたバイト数を返します。
タイムアウト無効（timeout=0）で本システムコールを呼び出すと、送信バッファに空きが無ければ直ぐに処理を中止し、*ercd に E_TMOUT を格納し、それまでに格納できたバイト数を返します。

8.4.10. 【関数名】TcpGetch

機能　ＴＣＰ受信バッファから１文字取得します。

書式　int TcpGetch(SOCKET s, long timeout, int *ercd);

引数		
	SOCKET s	ソケット・ディスクリプタ
	long timeout	タイムアウト設定（×１０ｍｓｅｃ）
	int *ercd	エラーコード格納先へのポインター

ＮＵＬＬ設定時は、エラーコードを返しません。

戻値　受信文字

【解説】

ＴＣＰ／ＩＰポートから受信バッファへの受信処理は、ＴＣＰ／ＩＰ受信タスクが行います。
本システムコールは、受信バッファから１文字を取得します。
受信バッファが空であった場合は、ＴＣＰ／ＩＰ受信タスクに受信要求を発行し、受信バッファにデータが格納されるまで待ち状態になります。
タイムアウト無し(timeout=-1)で本システムコールを呼び出した場合、受信が発生するまで待ち状態が続きます。
タイムアウト有り(timeout=1～0x7ffffff)で本システムコールを呼び出した場合は、指定した時間が経過してもデータを受信しなければ本システムコールは処理を中止し、*ercd に E_TMOUT を格納し、ＮＵＬ文字（０）を返します。
タイムアウト無効（timeout=0）で本システムコールを呼び出すと、受信バッファが空であった場合は直ぐに処理を中止し、*ercd に E_TMOUT を格納し、ＮＵＬ文字（０）を返します。

8.4.11. 【関数名】TcpIsRcv

機能　受信可能かどうかチェックします。

書式　int TcpIsRcv(SOCKET s);

引数　SOCKET s　　ソケット・ディスクリプタ

戻値　正の値　　受信データ有り
　　０　　受信データなし
　　負の値　　エラーコード
　　E_PAR　　ソケット・ディスクリプタの値が範囲外である.
　　E_SOCK_CLS　相手からの接続が正常に切断され、受信バッファにデータが無くなったことを示す(FIN 受信)
　　E_CLS　　相手からの接続が強制的に切断された(RST 受信)
　　その他標準エラーコード

8.4.12. 【関数名】TcpGets

機能　ＴＣＰ受信バッファから文字列を取得します。
　　改行文字を読み込むか、len-1 個の文字を読み込んだ場合に終了します。

書式　char *TcpGets(SOCKET s, char *buf, int len, int *bytes, char echo,
　　long timeout, int *ercd);

引数　SOCKET s　　ソケット・ディスクリプタ
　　char *buf　　読み込んだ文字列を格納するバッファへのポインター
　　int len　　バッファの長さ
　　int *bytes　　実際に読み込んだ文字数の格納先へのポインター

ＮＵＬＬ設定時は、文字数は返しません。

char echo　エコー設定

0　:エコーバックしません。

'*'　:'*'をエコーバックします

上記以外　:エコーバックをします。

long timeout　タイムアウト設定（×１０ｍｓｅｃ）

int *ercd　エラーコード格納先へのポインター

ＮＵＬＬ設定時は、エラーコードを返しません。

戻値　文字列に対するポインター（エラー時はＮＵＬＬ）

【解説】

本システムコールでは、１文字受信にTcpGetchシステムコール、エコーバックにTcpPutchシステムコールを使用しています。

タイムアウトは本システムコール全体の動作ではなく、TcpGetch システムコール、TcpPutch システムコールに対して設定されます。

各システムコールでタイムアウト(E_TMOUT)が発生した場合は、本システムコールは処理を中止し、*bytesにそれまでに取得できた文字数、*ercdにE_TMOUTを格納し、NULLを返します。

8.4.13. 【関数名】TcpPutch

機能　ＴＣＰ送信バッファに１文字格納します。

TcpPutch()では、ＴＣＰ送信バッファに空きがある場合は、ＴＣＰ送信バッファに書き込むだけで送信処理は行いません。

ＴＣＰ送信バッファ中のデータを、強制的に送信する場合は、TcpPush()を使用して下さい。

書式　void TcpPutch(SOCKET s, char c, long timeout, int *ercd);

引数　SOCKET s　ソケット・ディスクリプタ

char c　送信文字

long timeout　タイムアウト設定（×１０ｍｓｅｃ）

int *ercd　エラーコード格納先へのポインター

ＮＵＬＬ設定時は、エラーコードを返しません。

戻値　なし

【解説】

送信バッファからＴＣＰ／ＩＰポートへの送信処理は、ＴＣＰ／ＩＰ送信タスクが行います。

本システムコールは、送信バッファへ１文字格納します。

送信バッファが一杯であった場合は、ＴＣＰ／ＩＰ送信タスクに送信要求を発行し、送信バッファに空きが生じるまで待ち状態になります。

タイムアウト無し(timeout=-1)で本システムコールを呼び出した場合、送信バッファに空きが生じるまで待ち状態が続きます。

タイムアウト有り(timeout=1～0x7ffffff)で本システムコールを呼び出した場合、指定した時間が経過しても送信バッファに空きが生じなければ本システムコールは処理を中止し、*ercd に E_TMOUT を格納します。
タイムアウト無効（timeout=0）で本システムコールを呼び出すと、送信バッファに空きが無ければ直ぐに処理を中止し、*ercd に E_TMOUT を格納します。

8.4.14. 【関数名】TcpPuts

機能　ＴＣＰ送信バッファへ文字列を格納します。
TcpPuts()では、ＴＣＰ送信バッファに空きがある場合は、ＴＣＰ送信バッファに書き込むだけで送信処理は行いません。
ＴＣＰ送信バッファ中のデータを、強制的に送信する場合は、TcpPush()を使用して下さい。

書式　void TcpPuts(SOCKET s, char *str, int *bytes, long timeout, int *ercd);

引数		
	SOCKET s	ソケット・ディスクリプタ
	char *str	送信文字列
	int *bytes	実際に格納できた文字数へのポインター ＮＵＬＬ設定時は、文字数は返しません。
	long timeout	タイムアウト設定（×１０ｍｓｅｃ）
	int *ercd	エラーコード格納先へのポインター ＮＵＬＬ設定時は、エラーコードを返しません。

戻値　なし

【解説】
送信バッファからＴＣＰ／ＩＰポートへの送信処理は、ＴＣＰ／ＩＰ送信タスクが行います。
本システムコールは、指定された文字列を送信バッファへ格納します。
本システムコール処理中に送信バッファが一杯になった場合は、ＴＣＰ／ＩＰ送信タスクに送信要求を発行し、送信バッファに空きが生じるまで待ち状態になります。
タイムアウトは本システムコールの動作全体ではなく、送信バッファの空き待ちに対して設定されます。
タイムアウト無し(timeout=-1)で本システムコールを呼び出した場合、送信バッファに空きが生じるまで待ち状態が続きます。
タイムアウト有り(timeout=1～0x7ffffff)で本システムコールを呼び出した場合、指定した時間が経過しても送信バッファに空きが生じなければ本システムコールは処理を中止し、*bytes にそれまでに格納できた文字数、*ercd に E_TMOUT を格納します。
タイムアウト無効（timeout=0）で本システムコールを呼び出すと、送信バッファに空きが無ければ直ぐに処理を中止し、*bytes にそれまでに格納できた文字数、*ercd に E_TMOUT を格納します。

8.4.15. 【関数名】TcpPrintf

機能　書式付でＴＣＰ送信バッファへ文字列を格納します。

TcpPrintf()では、ＴＣＰ送信バッファに空きがある場合は、ＴＣＰ送信バッファに書き込むだけで送信処理は行いません。

ＴＣＰ送信バッファ中のデータを、強制的に送信する場合は、TcpPush()を使用して下さい。

書式　int TcpPrintf(SOCKET s, char *fmt, ...);

引数　SOCKET s　　ソケット・ディスクリプタ

　　char *fmt　　書式文字列（※１、※２、※３）

戻値　正常終了：出力文字数（≧０）

　　異常終了：エラーコード（＜０）

【解説】

書式変換後の文字列の送信は、TcpPutsシステムコールを使用します。

TcpPutsを呼び出す場合の、タイムアウト設定はタイムアウト無し(-1)で行います。

文字列送信動作の解説はTcpPutsシステムコールをご参照願います。

■注意事項

※１　フィールド幅あるいは精度に対する *指定はサポートしておりません。

※２　変換文字n(変換文字数の格納)はサポートしておりません。

※３　書式指定の詳細については、日立C/C++コンパイラ･マニュアルの「標準ライブラリ－fprintf関数」の項をご参照願います。

8.4.16. 【関数名】TcpPush

機能　送信バッファの内容を強制的に送信します。

書式　void TcpPush(SOCKET s, int *ercd);

引数　SOCKET s　　ソケット・ディスクリプタ

　　int *ercd　　エラーコード格納先へのポインター

　　ＮＵＬＬ設定時は、エラーコードを返しません。

8.4.17. 【関数名】TcpFlush

機能　送信バッファの内容を送信し、受信バッファを開放します。

書式　void TcpFlush(SOCKET s, int *ercd);

引数　SOCKET s　　ソケット・ディスクリプタ

　　int *ercd　　エラーコード格納先へのポインター

　　ＮＵＬＬ設定時は、エラーコードを返しません。

8.4.18. 【関数名】SetIpaddr

機能　IPアドレス及びサブネット マスクを設定します。

書式　void SetIpaddr(T_IPADDR *ipaddr);

引数　T_IPADDR *ipaddr　　IPｱﾄﾞﾚｽ及び SUBNET MASK 設定・取得用構造体へのポインター

```
typedef struct t_ipaddr {
    unsigned char ip_address[4];    // IPアドレス
    unsigned char subnet_mask[4];   // サブネット マスク
} T_IPADDR;
```

戻値　なし

■注意事項

1.本関数で設定した、IPアドレスは次回起動時から有効になります.
従って、本関数でIPアドレス設定後に、GetIpadd関数でIPアドレスを取得しても現在の設定値が返され、本関数で設定した値は返りません.

2.本関数は、装置のIPアドレスを直接変更します.
誤った設定を行うと、TCP/IP通信が出来なくなりますので十分注意してご使用願います。

3.本関数は、ＦＬＡＳＨメモリーの消去、書き換えを行います.
従って、関数実行中は決して、電源ＯＦＦやリセット当の操作をしないで下さい.

4.本関数は、ＦＬＡＳＨメモリーの消去、書き換えを伴う為、数秒程度時間がかかる場合があります。
この間、割込み動作は禁止していますので、本関数実行中は、通信やタイマー処理が発生しないようにしてください。
（通信の取りこぼし、タイマーの誤差等が発生する可能性が有ります）

■使用例

```
T_IPADDR ipaddr = {
    {192, 168,   0,  99},       // IP アドレス
    {255, 255, 255,   0}        // サブネット マスク
};

SetIpaddr(&ipaddr);
```

8.4.19. 【関数名】GetIpaddr

機能　現在設定されているIPアドレス及びサブネット マスクを取得します。

書式　void GetIpaddr(T_IPADDR *ipaddr);

引数　T_IPADDR *ipaddr　　IPｱﾄﾞﾚｽ及びSUBNET MASK設定・取得用構造体へのポインター

```
typedef struct t_ipaddr {
    unsigned char ip_address[4];    // IPアドレス
    unsigned char subnet_mask[4];   // サブネット マスク
```

```
    } T_IPADDR;
```

戻値　なし

1.本関数は、現在実際に設定されている IP アドレス及びサブネット マスクを返します.
　ＯＳモニターの ipadr コマンド、ipmsk コマンド及び SetIpaddr 関数で設定した IP アドレス及びサブネット マスクは次回起動時から有効になります.
　従って、本関数の返す IP アドレスも、次回起動時から設定された値になります.

2.デフォルト起動（「5.3　デフォルト起動」参）している場合は、設定に関係なく
　　IP アドレス　　= 192.168.0.99
　　サブネット マスク = 255.255.255.0
　として、起動します.
　従って、本関数も上記の IP アドレス及びサブネット マスクを返します.

8.5. グラフィック関連システムコール

液晶にグラフィック描画を行うシステムコール関数群です。
液晶座標はグラフィック座標を使用してください。(「4. LCD 表示とタッチパネルの概要」参)

8.5.1. 【関数名】DotCheck

機能 指定座標のドット情報を取得します。
書式 int DotCheck(int x, int y);
引数 int x,y グラフィック座標
戻値 0：ドットなし
1：ドット有り

8.5.2. 【関数名】DotSet

機能 点を描画します。
書式 void DotSet(int x, int y, int rop);
引数 int x,y グラフィック座標
int rop ラスターオペレーション
GCLR クリア
GSET セット
GXOR ＸＯＲ
戻値 なし

8.5.3. 【関数名】Line

機能 直線を描画します。
書式 void Line(short x1, short y1, short x2, short y2, int rop);
引数 short x1,y1 始点座標
short x2,y2 終点座標
int rop ラスターオペレーション
GCLR クリア
GSET セット
GXOR ＸＯＲ
戻値 なし

8.5.4. 【関数名】LineStyle

機能 線種を指定して直線を描画します。
書式 void LineStyle(short x1, short y1, short x2, short y2, int rop,
int width, short ptn);
引数 short x1,y1 始点座標

short x2,y2　　終点座標
int rop　　ラスターオペレーション
GCLR　クリア
GSET　セット
GXOR　ＸＯＲ　　(width == 1 以外では正常に動作しません)
int width　　線の太さ　　(1..15)
short ptn　　線種（例）0xffffで実線 0xf0f0で点線

戻値　なし

8.5.5.　【関数名】Circle

機能　円を描画します。

書式　void Circle(int x,int y, int r, int mode, int rop);

引数　int x,y　　円の中心座標
int r　　半径
int mode　　描画する部分
bit 0　　0:00 .. 1:30
bit 1　　1:30 .. 3:00
bit 2　　3:00 .. 4:30
bit 3　　4:30 .. 6:00
bit 4　　6:00 .. 7:30
bit 5　　7:30 .. 9:00
bit 6　　9:00 .. 10:30
bit 7　10:30 .. 12:00
int rop　　ラスターオペレーション
GCLR　クリア
GSET　セット
GXOR　ＸＯＲ

戻値　なし

8.5.6.　【関数名】BoxFill

機能　矩形領域の塗り潰しを行います。

書式　void BoxFill(short x1, short y1, short x2, short y2, int rop);

引数　short x1,y1　　左上座標
short x2,y2　　右下座標
int rop　　ラスターオペレーション
GCLR　クリア
GSET　セット
GXOR　ＸＯＲ

戻値　なし

8.5.7.　【関数名】PutImg

機能　画面イメージの設定を行います。

書式　void PutImg(int sx, int sy, int dx, int dy, char *img);

引数　int sx.sy　　画面イメージの左上座標

　　int dx,dy　　画面イメージの大きさ

　　char *img　　画面イメージのエリア

戻値　なし

8.5.8.　【関数名】PutImgOr

機能　画面イメージの設定を行います。ただし背景は消しません。

書式　void PutImgOr(int sx, int sy, int dx, int dy, char *img);

引数　int sx.sy　　画面イメージの左上座標

　　int dx,dy　　画面イメージの大きさ

　　char *img　　画面イメージのエリア

戻値　なし

8.5.9.　【関数名】GetImg

機能　画面イメージの取得を行います。

書式　void GetImg(int sx, int sy, int dx, int dy, char *img);

引数　int sx.sy　　画面イメージの左上座標

　　int dx,dy　　画面イメージの大きさ

　　char *img　　イメージを格納するエリア

　　戻値　なし

■注意事項

メモリーエリアの確保はユーザーが行って下さい。

画面イメージの取得に必要なメモリーエリアは(dx + 7) / 8 * dy バイトです。

8.5.10.　【関数名】PutWinbmp

機能　WINDOWSのモノクロビットマップを表示します。

書式　int PutWinbmp(int sx, int sy, char *fname, int rev);

引数　int sx,sy　　画像を張り付ける左上座標

　　char *bmpname　　ビットマップ名

　　int　rev　　反転表示フラグ（0で反転表示、1で通常表示）

戻値　エラーコード

　　E_OK　　成功

E_PAR	パラメータエラー
E_NOEXS	指定されたビットマップが無い
E_OBJ	指定されたビットマップが適切でない

■注意事項

表示するビットマップはあらかじめ BIN2S3.EXE を使用して、モトローラ S フォーマットに変換した上で NpTerm によりダウンロードしておいて下さい。

ファイル変換時の注意事項につきましては、「10.2　Windows ビットマップに関する注意事項」の項をご参照願います。

8.5.11. 【関数名】TextOut

機能　文字列をグラフィック表示します

書式　void TextOut(char *hnfont, char *znfont,int x, int y, int dx, int dy,
int mode,char *str);

引数	char *hnfont	半角フォントファイル名 NULL なら標準フォント使用。
	char *znfont	全角フォントファイル名 NULL なら標準フォント使用。
	int x,y	表示開始左上座標
	int dx,dy	表示文字の大きさを半角フォントのサイズで指定します。 dx=dy=0 を指定すれば字種の大きさで表示します。
	int mode	表示モード bit0　1:反転表示 bit1　1:上書き表示（背景を消さない）
	char *str	表示文字列

戻値　なし

■使用例

座標(0,0)から標準フォントを用い"HELLO"を字種の大きさ(16ﾄﾞｯﾄ)で表示

```
TextOut(NULL, NULL, 0, 0, 0, 0, 0, "HELLO");
```

座標(10,0)から標準フォントを用い"HELLO"を倍角で反転表示

```
TextOut(NULL, NULL, 10, 0, 16,32, 1, "HELLO");
```

座標(0,10)から半角フォントに"hn24.fnt", 全角フォントに"zn24.fnt"を用い"HELLO"を
字種の大きさで表示（あらかじめ FLASH メモリーに hn24.fnt zn24.fnt をダウンロードしておく必要があります)。

```
TextOut("hn24.fnt", "zn24.fnt", 0, 10, 0,0,0, "HELLO");
```

8.6. ＩＯポート関連システムコール

「J11　拡張 I/O コネクタ」に接続されている、拡張 I/O ポートを制御する関数群です。
１６本の I/O ポート（IO0～IO15）と、４本の A/D 入力ポート（AD0～AD3）があります。
IO0～IO15 は全て入力にも出力にも使用することが出来ます。
IO8，IO9 はシリアルポート（COM3）として使用することも出来ます。
IO10，IO11 はシリアルポート（COM4）として使用することも出来ます。
IO12，IO13 は HCAN（HCAN1）として使用することも出来ます。
IO14，IO15 は HCAN（HCAN2）として使用することも出来ます。
ポートの初期設定は IniPort 関数を使用して行います。

8.6.1. 【関数名】IniPort

機能　拡張ポートの初期設定

書式　void IniPort(unsigned short dir, int com3_use, int com4_use, int hcan1_use,
int hcan2_use);

引数　unsigned short dir　ポートの方向を設定します(0:IN PORT, 1:OUT PORT)
BIT0=DIR0…BIT15=DIR15 とﾋﾞｯﾄで割り付けてあります。
int com3_use　0 : IO8，IO9 を汎用ポートとして設定します。
この場合は DIR によって入出力の方向を指定します
1 : IO8=COM3_TXD，IO9=COM3_RXD として設定します。
この場合は DIR の設定は無視されます。
int com4_use　0 : IO10，IO11 を汎用ポートとして設定します。
この場合は DIR によって入出力の方向を指定します
1 : IO10=COM4_TXD，IO11=COM4_RXD として設定します。
この場合は DIR の設定は無視されます。
int hcan1_use　0 : IO12，IO13 を汎用ポートとして設定します。
この場合は DIR によって入出力の方向を指定します
1 : IO12=HCAN1_TXD，IO11=HNAC1_RXD として設定します。
この場合は DIR の設定は無視されます。
int hcan2_use　0 : IO14，IO15 を汎用ポートとして設定します。
この場合は DIR によって入出力の方向を指定します
1 : IO14=HCAN2_TXD，IO15=HNAC2_RXD として設定します。
この場合は DIR の設定は無視されます。

戻値　なし

8.6.2. 【関数名】OutPort

機能　拡張ポートに出力をします。
通信専用に割り当てられたピンについては、ポート内のデータレジスタに設定され

ますが、端子(ピン)には出力されません。

書式　void OutPort(unsigned short pd);

引数　unsigned short pd　出力データ

戻値　なし

8.6.3.　【関数名】InPort

機能　拡張ポートから入力します。

通信専用に割り当てられたピンについても、端子(ピン)の状態が帰ります。

書式　unsigned short InPort(void);

戻値　ポートデータ

8.6.4.　【関数名】InAD

機能　ＡＤ変換された値を取得する

書式　unsigned short InAD(int ad_num);

引数　int ad_num　　取得するＡＤ番号

戻値　正常時　　ＡＤ変換された値（０～１０２４）

エラー時　－１

■注意事項

入力可能電圧は０Ｖ～５Ｖまでです。

9. 利用可能マクロ

9.1. 組込みタイマー

9.1.1. TIMER_L

システム起動時にリセットされ、1ミリ秒毎にカウントアップされる３２ビットのタイマー変数です。
約 50 日間弱のカウントが可能です。(読み書き可能)

9.1.2. TIMER_1

０になるまで１ミリ秒毎にカウントダウンされる３２ビットのタイマー変数です。(読み書き可能)

9.1.3. TIMER_2

０になるまで１ミリ秒毎にカウントダウンされる３２ビットのタイマー変数です。(読み書き可能)

9.1.4. TIMER_SEC

システム起動時にリセットされ、ＲＴＣ（リアルタイムクロック）により、１秒毎にカウントアップされる３２ビットのタイマー変数です。(読み書き可能)

10. 注意事項

10.1. ファイル送信に関する注意事項

10.1.1. ファイル送信について

NP2000では、ファイルシステムを実装しておりません、本書で説明している「ファイル送信」とはNpTerm側からモトローラＳフォーマットのファイルを送信し、NP2000側は受信したフォーマットを解析し指定されたアドレスへデータを格納することを意味します。

10.1.2. ファイル（データ）格納領域

ファイル(データ)を格納できる領域は、NP2000に実装された、1MバイトのFLASHメモリ領域です。
FLASHメモリのアドレス及び状態は「ERASE INFO」コマンドで確認することが出来ます。

「ERASE INFO」コマンド実行例

```
>ERASE INFO
Flash[0]:Sector[0H]:adr=00800000H:Erase NG --- APPLICATION
Flash[0]:Sector[2H]:adr=00820000H:Erase OK --- APPLICATION
Flash[0]:Sector[4H]:adr=00840000H:Erase OK --- APPLICATION
Flash[0]:Sector[6H]:adr=00860000H:Erase NG --- USER
Flash[0]:Sector[8H]:adr=00880000H:Erase NG --- FONT
Flash[0]:Sector[aH]:adr=008a0000H:Erase NG --- FONT
Flash[0]:Sector[cH]:adr=008c0000H:Erase OK --- RESERVE
Flash[0]:Sector[eH]:adr=008e0000H:Erase NG --- SYSTEM
-- DOWNLOAD INFOMATION --
  SAMPLE1     0x00800000 - 0x00803C3C  NP2000 APPLICATION     [CRC OK]
```

※ adr=00800000Hの部分が、物理アドレスを示します。
※ 「Erase OK」と表示された領域は、Flash メモリが消去済みでデータは格納されていないことを示します。
※ 「Erase NG」と表示された領域は、Flash メモリに何らかのデータが格納されていることを示します。
※ 「APPLICATION」と表示された領域は、アプリケーション用に解放された領域で、「アプリケーションプログラム」、「Windowsビットマップ」、「その他アプリケーションに必要なデータ」を格納するための領域です。
※ 「USER」と表示された領域は、BackupDataSave関数により、バックアップデータを書き込むために確保されている領域です。データの読み出しにはBackupDataLoad関数を使用します。
※ 「FONT」と表示された領域は、システムフォントを格納する領域です。
※ 「RESERVE」と表示された領域は、NP2000-OSの予約領域です。
※ 「SYSTEM」と表示された領域は、NP2000-OSのシステム設定等を格納する領域です。
※ 「DOWNLOAD INFORMATION」以降には現在、「ファイル送信」を使用してダウンロードされているデ

ータの一覧を表示しています。

ユーザーが「ファイル送信」を使用して、データをダウンロードできる領域は、APPLICATION 領域のみです。その他の領域は、NP2000-OS で管理されていますので、ここにデータがダウンロードされるようなファイルは送信しないで下さい。

10.1.3. ファイル(データ)格納領域の消去

前項にも記したとおり、ダウンロード領域は Flash メモリです。
従って、ダウンロードする領域は予め消去されている必要がありますが、NP2000 側ではダウンロード領域を自動的に消去する機能は持たせていません。
この為、「ファイル送信」先立って、「ERASE　APP」コマンドでアプリケーション領域を消去しておく必要があります。

10.1.4. ダウンロード情報

OS モニターの、「DLINFO」コマンドを使用して、現在 Flash メモリにダウンロードされているデータの一覧を見ることが出来ます。

「DLINFO」コマンド実行例

```
>DLINFO
  SAMPLE1     0x00800000 - 0x00803C3C  NP2000 APPLICATION     [CRC OK]
```

また、前述しているように、「ERASE　INFO」コマンドを実行した場合にも、末尾にダウンロード情報が表示されます。

10.1.5. ファイル比較

「ERASE APP」コマンドを実行せずに、「ファイル送信」を実行した場合で、ダウンロード情報の中に送信しようとしているファイルと同じ名前のデータが存在する場合は、NP2000 はダウンロードデータの書き込みを実施せずダウンロードされてきたデータと Flash メモリ内に存在する同名データとの比較を行います。

10.1.6. 送信ファイルの作成

「ファイル送信」できるファイルは、モトローラＳフォーマットのファイルのみです。
アプリケーションプログラムについては、コンパイル・リンク後にコンパイラ付属のフォーマット変換ユーティリティーでモトローラＳフォーマットを作成してください。
変換方法につきましては、「7.6.5　サンプルプログラムのファイル変換」の項をご参照願います。
その他の、ファイルについては、汎用のフォーマット変換ユーティリティーBIN2S3.EXE を付属しておりますのでそれを用いてフォーマットを変換後、「ファイル送信」を行います。
BIN2S3 につきましては「BIN2S3 操作説明書」をご参照願います。

※　複数のデータをダウンロードすることは可能ですが、その場合各ファイル(データ)のアドレスが重ならないように十分注意して、送信ファイルを作成して下さい。

10.2. Windows ビットマップに関する注意事項

NP2000 では、WINBMP コマンドまたは、PutWinbmp 関数により、WINDOWS のモノクロビットマップを表示することが出来ます。
このためには、あらかじめ NpTerm を使用して「ファイル送信」が出来るように、モトローラＳ形式のファイルに変換する必要があります。
ファイル変換には、BIN2S3.EXE を使用しますが、Windows ビットマップに関しては変換に際して以下の制限がありますのでご注意願います。

● Windowsモノクロビットマップ→モトローラＳ形式変換時の制限

変換時の開始アドレスの末尾は 2,6,A,C のいずれかである必要があります。

● 解説

Windows モノクロビットマップの構造体は、Intel 系の CPU に対応しています。
SH7055 では、ワード(16 ﾋﾞｯﾄ)のデータ形式は 2n 番地、ロングワード(32 ﾋﾞｯﾄ)のデータは 4n 番地から始まる必要があるというアライメント上の制限があるため、Windows モノクロビットマップの構造体を扱うためには、データの先頭番地が４の倍数でない２の倍数である必要があります。
従って、変換時は開始アドレスを 2×m(m=1,3,5,7)、つまりアドレスの末尾１バイトを２、６、Ａ、Ｃのいずれかにして下さい。

● 変換例

以下の例では、NP2000.BMP を変換開始番地 = 0x00878002 番地として、NP2000.MOT に変換します。

```
BIN2S3 NP2000.BMP 0x00878002 > NP2000.MOT
```

この、NP2000.MOT を NpTerm の「ファイル送信」機能を使用して、NP2000 にダウンロードします。

10.3. ＯＳアップデート回数の制限

NP2000 で使用している CPU（SH7055）の内蔵 FLASH メモリーへの書き込み回数の保障値は 100 回です。
従って、不必要な OS ダウンロードはしないようにして下さい。